STORAGE MANAGEMENT AND ACCESS IN WLHC COMPUTING GRID

by

Dr. Flavia Donno

A thesis submitted in partial fulfillment of the requirements for the degree of

PhD in Information Engineering

Curriculum: Architecture of Computing Systems

University of Pisa

2005-2006

Approved by Prof. Gigliola Vaglini _____
Dr. Andrea Domenici _____

Date _____

UNIVERSITY OF PISA

ABSTRACT

STORAGE MANAGEMENT AND
ACCESS IN LHC COMPUTING GRID

by Dr. Flavia Donno

Supervisors: ***Prof. Gigliola Vaglini***
Dr. Andrea Domenici
Department of Computer Engineering

One of the big challenges in Grid computing is storage management and access. Several solutions exist to store data in a persistent way. In this work we describe our contribution within the Worldwide LHC Computing Grid project. Substantial samples of data produced by the High Energy Physics detectors at CERN are shipped for initial processing to specific large computing centers worldwide. Such centers are normally able to provide persistent storage for tens of Petabytes of data mostly on tapes. Special physics applications are used to refine and filter the data after spooling the required files from tape to disk. At smaller geographically dispersed centers, physicists perform the analysis of such data stored on disk-only caches. In this thesis we analyze the application requirements such as uniform storage management, quality of storage, POSIX-like file access, performance, etc. Furthermore, security, policy enforcement, monitoring, and accounting need to be addressed carefully in a Grid environment. We then make a survey of the multitude of storage products deployed in the WLCG infrastructure, both hardware and software. We outline the specific features, functionalities and diverse interfaces offered to users. Among the other storage services, we describe StoRM, a storage resource manager that we have designed and developed to provide an answer to specific user request for a fast and efficient Grid interface to available parallel file systems. We propose a model for the Storage Resource Management protocol for uniform storage management and access in the Grid. The black box testing methodology has been applied in order to verify the completeness of the specifications and validate the existent implementations. We finally describe and report on the results obtained.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I wish to express sincere appreciation to Prof. Gigliola Vaglini and Dr. Andrea Domenici for their assistance in the preparation of this manuscript and for their supervision during my PhD studies. In addition, special thanks to Prof. Mirco Mazzucato and Dr. Antonia Ghiselli from INFN (Italian National Institute of Nuclear Physics) who made possible my participation to several Grid projects. I am especially grateful to the following people who are a big source of inspiration and from whom I learned everything I know about SRM and much more: Jean-Philippe Baud and Maarten Litmaath from CERN (Switzerland), Arie Shoshani, Alex Sim and Junmin Gu from LBNL (U.S.), Timur Perelmutov from FNAL (U.S.). I can never stop admiring the code written by Jirí Mencák from RAL (UK) for the S2 interpreter: it is really a must for all students approaching object oriented and C++ programming. A big thank you to the colleagues of the StoRM project from INFN and ICTP (Luca Magnoni, Riccardo Zappi, Ezio Corso and Riccardo Murri) for the many discussions and for sharing with me the difficult steps involved in the design and development of a software product for a production Grid environment. Finally I would like to thank my bosses at CERN, Les Robertson, Ian Bird and Jamie Shiers who always encouraged and supported me in the difficult task of "making things work"!

A special thank goes to the father of my son, Heinz, who has shared with me the difficult moments and has strongly supported me discussing many technical issues, while providing Kilian with many "Unterhosenservices". A big hug and a huge "Bussi" to my little son Kilian, who allowed me to work on my PhD project during the first months of his life.

Finally, I would like to thank my mother and my father who have always been by my side, encouraging me in pursuing this career and giving me the possibility to live this wonderful and exciting experience.

GLOSSARY

AC Attribute Certificate

ACL Access Control List

AFS Andrew File System

API Application Programming Interface

ARC Advanced Resources Connector

BDII Berkeley Database Information Index

BESTMAN Berkeley Storage Manager

CE Computing Element

CA Certification Authority

CAS Community Authorization Server

CASTOR CERN Advanced STORage Manager

CERN Conseil Européen pour la Recherche Nucléaire

CLI Command Language Interface

DAP Data Access Point

DAS Direct Attached Storage

DCACHE Disk Cache

DESY Deutsches Elektronen-SYnchroton

DIT Directory Information Tree

DLI Data Location Interface

DM Data Management

DMF Data Migration Facility

DN Distinguished Name

DPM Disk Pool Manager

DRM Disk Resource Manager

DRS Data Replication Service

DST Distributed Storage Tank

EDG European Data Grid

EGEE Enabling Grid for E-SciencE

FNAL Fermi National Accelerator Laboratory

FQAN Fully Qualified Attribute Name

FTS File Transfer Service

GAA Generic Authorization and Access

GASS Global Access to Secondary Storage

GDMP Grid Data Mirroring Package

GFAL Grid File Access Library

GG Grid Gate

GIIS Grid Information Index Service

GLUE Grid Laboratory for a Uniform Environment

GOC Grid Operation Center

G-PBOX Grid Policy Box

GPFS General Parallel File System

GRAM Globus Resource Allocation and Management

GRIP Grid Resource Information Protocol

GRIS Grid Resource Information Service

GSI Grid Security Infrastructure

GSM-WG Grid Storage Management – Working Group

GSS-API Generic Security Service Application Programming Interface

GT Globus Toolkit

GUID Grid Unique IDentifier

HEP High Energy Physics

HPSS High Performance Storage System

ICTP Abdus Salam International Center for Theoretical Physics

INFN Istituto Nazionale Fisica Nucleare

IS Information Service

JDL Job Description Language

LB Logging and Bookkeeping Service

LBNL Lawrence Berkeley National Laboratory

LCAS Local Centre Authorization Service

LCG LHC Computing Grid

LCMAPS Local Credential Mapping Service

LDAP Lightweight Directory Access Protocol

LDPM LCG Disk Pool Manager

LHC Large Hadrons Collider

LFC LCG File Catalogue

LFN Logical File Name

LOA Legion Object Address

LOID Legion Object IDentifier

LSF Load Sharing Facility

LRMS Local Resource Management System

LRU Least Recently Used

MDS Monitoring and Discovery Service

MSS Mass Storage System

NAS Network Attached Storage

NDGF Nordic Data Grid Facility

NFS Network File System

OGF Open Grid Forum

OGSA Open Grid Service Architecture

OPR Object Persistent Representation

OSG Open Science Grid

PAT Policy Administration Tool

PDP Policy Decision Point

PEP Policy Enforcement Point

PKI Public Key Infrastructure

PR Policy Repository

PVFS Parallel Virtual File System

RB Resource Broker

RFIO Raw File I/O

RFTS Reliable File Transfer Service

RGMA Relational Grid Monitoring Architecture

RLS Replica Location Service

RSL Resource Specification Language

SAM Storage Access Manager

SAN Storage Area Network

SE Storage Element

SOAP Simple Object Access Protocol (original meaning)

SQL Structured Query Language

SRB Storage Resource Broker

SRM Storage Resource Manager

SSE Smart Storage Element

STORM STOrage Resource Manager

SURL Storage URL

SUT System Under Test

TDR Technical Design Report

TLS Transport Layer Security

TSM Tivoli Storage Manager

TURL Transport URL

UI User Interface

VDC Virtual Data Catalog

VDL Virtual Data Language

VDT Virtual Data Toolkit

VOMS Virtual Organization Management System

WLCG Worldwide LHC Computing Grid

WMS Workload Management System

WN Worker Node

WSDL Web Service Description Language

WSRF Web Service Resource Framework

PREFACE

Grid Computing is one of the emerging research fields in computer science. The Grid aims at providing an infrastructure that enables the sharing, selection, and aggregation of geographically distributed "autonomous" resources dynamically depending on their availability, capability, performance, cost, and users' quality-of-service requirements. Users belonging to a "Virtual Organization" can establish policies of usage, requirements, a working environment and even a set of virtual resources for operation. From the time of the first proposal made by Ian Foster and Carl Kesselman with the publication of the book "The Grid: Blueprint for a new computing infrastructure" and the development of the Globus Toolkit, the Grid has gone through major evolutions, attracting industry partners as well. In Europe, the projects European DataGrid (EDG), Worldwide Large Hadrons Collider Computing Grid (WLCG) and Enabling Grid for E-SciencE (EGEE) have promoted the development of Grid middleware and the creation of a worldwide computing infrastructure available for science and research.

Even though current middleware solutions are much more complete than the first prototype proposed by the Globus Toolkit, there are many areas that still need investigations and development. One of these is certainly storage management and access. There are many research challenges: applications running on the Grid need to transparently access data on the specific local storage device, exploiting a set of needed features such as space and quota management, POSIX file access, security and policy enforcement, reliable file movement, without being aware of the specific hardware/software solutions implemented at a site.

At the time of writing a complete, self-contained and coherent solution to storage management is missing in many of the existing Grid research infrastructures today: WLCG in Europe, Nordic Data Grid Facility (NDGF) in the Northern European countries, Open Science Grid (OSG) in USA, etc. One of the issues that complicate the task is the heterogeneity of storage solutions used in computing centers around the world. This work aims at providing a proposal for v2.2 Storage Resource Manager (SRM) protocol, a Grid protocol for storage systems that provides for uniform storage management capabilities and flexible file access. In particular, a formal model has been designed and applied to check the consistency of the specification proposed. The test modeling approach has been used to generate a test suite to validate the implementations made available for the storage services deployed in the WLCG infrastructure. The study

of the black box testing methodology applied to SRM has allowed us to find many inconsistencies in the specifications and to deeply test the behavior of several SRM implementations. In order to converge toward a first "real" implementation of SRM in version 2 we left uncovered issues and features that will be dealt with in version 3 of the SRM protocol. The first deployment in production of SRM v2.2 based storage services is foreseen in June 2007. Among the solutions, which are SRM v2.2 based, there are CASTOR developed at CERN and Rutherford Appleton Laboratory (RAL), dCache developed at Deutsches Elektronen-Synchroton (DESY) and Fermi National Accelerator Laboratory (FNAL), LDPM developed at CERN, BeStMan developed at LBNL and StoRM developed in Italy by Istituto Nazionale di Fisica Nucleare (INFN) and International Centre of Theoretical Physics (ICTP). StoRM is a disk-based storage resource manager designed to work over native parallel filesystems. It provides for space reservation capabilities and uses native high performing POSIX I/O calls for file access. StoRM takes advantage of special features provided by the underlying filesystem like ACL support and file system block pre-allocation. Permission management functions have also been implemented. They are based on the Virtual Organization Management System (VOMS) and on the Grid Policy Box Service (G-PBox). StoRM caters for the interests of the economics and finance sectors since security is an important driving requirement.

Chapter 1

INTRODUCTION

The vision of Grid computing was introduced by Ian Forster and Carl Kesselman with the publication of their book "The Grid: Blueprint for a New Computing Infrastructure" in July 1998 [1]. The idea is to virtualize computing, with the goal of creating a utility computing model over a distributed set of resources. Figure 1.1 helps to visualize the concepts.

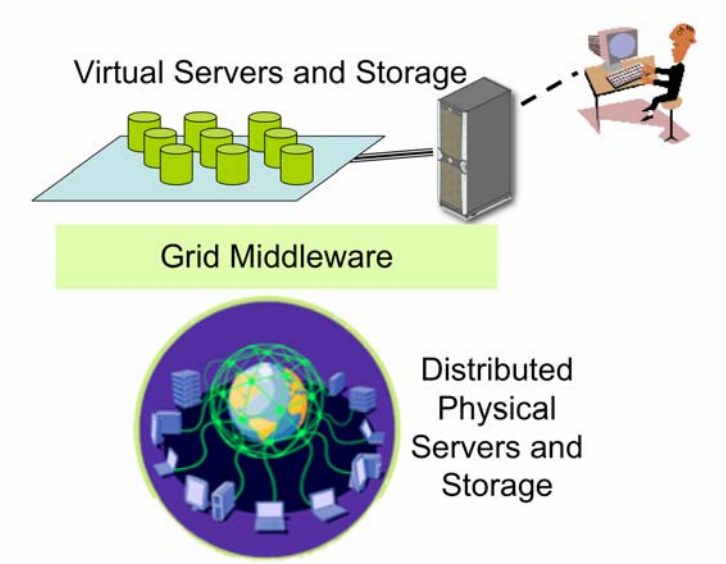


Figure. 1.1 Virtualization of computing and storage resources

Within a single computer, standard elements including the processor, storage, operating system, and I/O exist. The concept of Grid computing is to create a similar environment, over a distributed area, made of heterogeneous elements including servers, storage devices, and networks – a scalable, wide-area computing platform.

*The **Grid Middleware** is the software that handles the coordination of the participating elements.* It is analogous to the operating system of a computer.

A ***Grid Service*** *is a special service that contributes to make the Grid infrastructure available to users.* It is analogous to an operating system component such as the filesystem or the memory manager.

The Grid middleware is organized in a "Grid service" layered stack for performing the various operations. The Grid services provide users with standard and uniform interfaces over all the sites participating to the Grid.

A ***Grid Resource*** *is component of the system that provides or hosts services and may enforce access to these services based on a set of rules and policies defined by entities that are authoritative for the particular resource.*

Typical resources in Grid environments might be a computer providing compute cycles or data storage through a set of services it offers. Access to resources may be enforced by a Resource itself or by some entity (a policy enforcement point, gateway) that is located between a resource and the requestor thus protecting the resource from being accessed in an unauthorized fashion.

No matter what the hardware and software solutions used to create a Grid participating site are, users can always use a transparent and standard interface to access the available resources. Also many different Grid services can be used for performing the same task. Such Grid services publish their interfaces that can be therefore used by other Grid services as well as by higher-level user applications.

1.1 Problem Statement

As it happens on a local computer, application specific persistent data and metadata in a Grid are stored on disk or tape systems. Whether running on a LAN cluster or over a Grid infrastructure, a computational user task often needs to access data in order to calculate the expected result. The output of such a task might be of interest to other communities of users or can be used for further processing. No matter what the underlying storage technology is, a certain number of functionalities need to be guaranteed to an application. First of all the availability of input data must be guaranteed to the application. Such data set must stay on the device accessed by the application for the entire life of the job accessing it.

- Mechanisms of ***file pinning*** must guarantee that a storage garbage collector does not remove files used by the application.

The application must be enabled to use the data access protocol encoded in the application to access the data.

- The storage device offering the data has to support such ***access protocols.***
- ***Authorization and security policies*** (local or global) need to be enforced during data access.

Before moving jobs that generate large output files to Grid computing systems with available CPU capacity, a Grid scheduler should check for availability of the required space and then allocate it.

- ***Space reservation*** is another important requirement especially when using the Grid.
- The possibility to ***manage disk-space*** via the Grid becomes essential for running data-intensive applications on the Grid.
- ***Data movement and replication*** are also important functions to guarantee optimized access to files.
- Finally, services that guarantee consistency of modifiable files need storage services offering features such as ***file locking***.

1.2 Contribution of this Thesis

This thesis has been carried out at CERN within the WLCG project. The aim of the project is to provide the four LHC experiments at CERN with a production ready Grid infrastructure distributed all over the

world that allows for the storage, the processing and the analysis of data produced by the four detectors of physics particle collisions positioned on the Large Hadrons Collider (LHC) accelerator running at CERN.

Each of the four HEP experiments registers collision events generated by the particles accelerated by the LHC. The raw data generated can be of several Petabytes for each of the experiments. Each member of the worldwide-distributed collaboration of physicists needs to have efficient, easy, and transparent access to the data in order to analyze them and contribute to the physics discovery. Therefore, data management and storage access are important factors in the WLCG Grid infrastructure.

Among the original contributions of this thesis work we list the following:

1. A comprehensive overview of the state of art for what concerns storage management in the Grid, open issues and current developments.
2. A study of the Storage Resource Management (SRM) interface, as the attempt to define and propose a common interface to the diverse storage solutions adopted by the distributed computing centers.
3. A formal model of the SRM protocol as defined by the interface as the set of ordered interactions between client and server.
4. The proposal of a model for a schema to be used to publish the information related to the SRM based storage services.
5. The application of the testing black box methodology to the SRM in order to find out incompleteness and incoherence in the specification.
6. A study on how to reduce the number of tests to be designed to validate the protocol implementations.
7. A presentation and analysis of the results obtained during the test phase.
8. A discussion on the limitations and open issues of the current available version of SRM.
9. A proposal for a new version of the SRM protocol implementing the need for quota management and lock functions besides solving the intrinsic problems posed by the current versions.

1.3 Outline

In this work a comprehensive introduction to Grid computing pointing out the key aspects, general components and existing implementations can be found in Chapter 2. In Chapter 3 we give an introduction to the storage problem in the Grid outlining the requirements for LHC experiments. In Chapter 4, we examine some of the existing storage solutions proposed by the different vendors from hardware and software perspectives and their characteristics interesting for implementing Grid solutions.
In particular, we give an overview of existing distributed and parallel file systems and we described StoRM, a disk based storage system based on parallel filesystems such as GPFS. StoRM has been used to initially verify the feasibility of the proposed SRM protocol. In Chapter 5 we give an overview of the file access and transfer protocols used by HEP applications running on the WLCG infrastructure. We also introduce the GridFTP protocol, one of the very first transfer protocols Grid enabled that is in wide use today in the Grid infrastructure for e-Science. In Chapter 6 we analyze the Storage Resource Manager (SRM) version 2.2, an attempt to standardize storage management and access in the Grid. In particular, we define a formal model for the protocol behind the interface. In Chapter 7, we propose a model and a schema to publish in the Grid the information related to an SRM based storage service. We introduce new concepts such as Storage Areas and Components that will be exposed only in version 3 of the SRM

protocol. In Chapter 8 we present the application of the black box testing methodology to SRM to check the consistency and coherency of the specification and validate existing implementations. The result are also presented and analyzed together with some notes on the practical lessons learned. Finally, in Chapter 9 we give some conclusive comments. We discuss the limitations and the open issues present in version 2.2 of SRM and we introduce version 3, currently under definition. We also give an overview of possible future work in this area in order to achieve a completely functional storage solution for Grid.

2. GRID COMPUTING AND ARCHITECTURE

In this chapter we illustrate Grid computing and provide an overview of its architecture and components. In the academic as well as in the commercial worlds there are several definitions of Grid computing [2]. However, they all focus on the need of virtualizing a set of distributed resources in order to provide a scaleable, flexible pool of processing and data storage that can be used to improve efficiency. Another key of Grid computing is the promotion of standards that allow for interoperability of systems provided by many vendors and modular integration to create complex systems. Grid computing helps create a sustainable competitive advantage by way of streamlining product development and allowing focus to be placed on the core business.

The ***Grid computing market*** is in a relatively early stage [3][9], but now it is the time to initiate Grid-related developments for several reasons, particularly the following ones:

- The need for communication and efficient interconnection is becoming important in order to have predominance in the market.
- Emerging applications are significant, coming from increasingly important markets, such as energy and oil, financial services, government, life sciences, and manufacturing (sources: IDC, 2000 and Bear Stearns- Internet 3.0 - 5/01 Analysis by SAI).
- The infrastructure to support these applications is currently underserved.
- The potential market size is substantial (including hardware and software associated with grid deployments).
- Investment commitment and development focus from the industry's largest computing players, including HP, IBM, Microsoft, and Sun, is an indicator that this is a growth market.
- Increased deployment of blade servers coincides with the related view of blade server vendors that clusters and Grids are ways of moving forward.
- There is increasing pressure for enterprise IT organizations to cut costs and increase utilization of existing infrastructures.

Web services *are distributed software components that provide information to applications, through an application-oriented interface.*

Grid environments enable the creation of virtual organizations (defined later in this chapter) and advanced Web services.

Referring to Grid computing as "The Grid," is not necessarily the most appropriate or accurate reference. Although it is convenient for introducing a high-level discussion, there is not one single "Grid". Instead, there are many Grids, some private, some public, some distributed worldwide and even some local to one room.

In the second part of this chapter we will provide an overview of existing Grid software systems and solutions, pointing out advantages and disadvantages, maturity and deployment.

2.1 Grid Computing

In the following section we give some fundamental definitions in order to provide a common knowledge base for this thesis. There are many *definitions* for Grid computing. The following three are quite comprehensive:

- The Grid is an aggregation of geographically dispersed computing, storage, and network resources, coordinated to deliver improved performance, higher quality of service, better utilization, and easier access to data.
- The Grid enables virtual, collaborative organizations, sharing applications, resources and data in an open, heterogeneous environment. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully what is shared, who is allowed to share, and the conditions under which sharing occurs.
- The Grid promotes standard interfaces definitions for services that need to inter-operate to create a general distributed infrastructure to fulfill user's tasks and provide user level utilities.

The common denominator for all Grid infrastructures is the network layer. Since it is this common network fabric that connects all of the resources in a given Grid, its significance is amplified. Distributed Grid systems demand high-speed connectivity and low latency.

A ***Grid application*** *can be defined as an application that operates in a Grid environment.*

From an application perspective, there are two types of Grids: ***Computing Grids*** and ***Data Grids***. The majority of the early Grid deployments has focused on computation, but as Data Grids provide easier access to large, shared data sets, Data Grids are becoming more and more important.

A ***Computing Grid*** [4] is a collection of distributed computing resources, within or across sites, which are aggregated to act as a unified processing virtual supercomputer. These compute resources can be either within or across administrative domains. Collecting these resources into a unified pool involves coordinated usage policies, job scheduling and queuing characteristics, Grid-wide security, and user authentication. The benefit is faster, more efficient processing of compute-intensive jobs, while utilizing existing resources.

Computing Grids have typically higher latencies than clusters. However, they provide for more computing power due to the CPU aggregation. Compute Grids also eliminate the drawback of tightly binding specific machines to specific jobs, by allowing the aggregated pool to most efficiently serve sequential or parallel jobs with specific user requirements.

A ***Data Grid*** [5] provides for wide-area, secure access to significant amount of data. Data Grids enable the management and efficient usage of data stored in different formats as well as in distributed locations. Much like Computing Grids, Data Grids also rely on software for secure access and usage policies. However Grid storage solutions assume a quite important role in this context. Data Grids can be deployed within one administrative domain or across multiple domains. In such cases Grid software and policy management become critical. Data Grids reduce the need to move, replicate, or centralize data, translating into cost savings. Initial Data Grids are being constructed today, primarily serving collaborative research communities [16,17,18,19]. Software vendors and large enterprises are currently investigating Data Grid solutions and services for business applications [9].
The evolution from Computing Grids to Data Grids is an important factor in moving Grid technology from education and R&D to the large enterprise. This transition is an indicator that the market, in

addition to the technology, is maturing. From a networking perspective, the impact of Data Grids will include a tighter integration of storage protocols and high-performance networking.

The first stage of Grid computing is a ***cluster*** [6] (cf. Figure 2.1). Because the most common definition of a Grid includes terms like "distributed" and "heterogeneous," it is debatable whether clusters should actually be considered Grids.

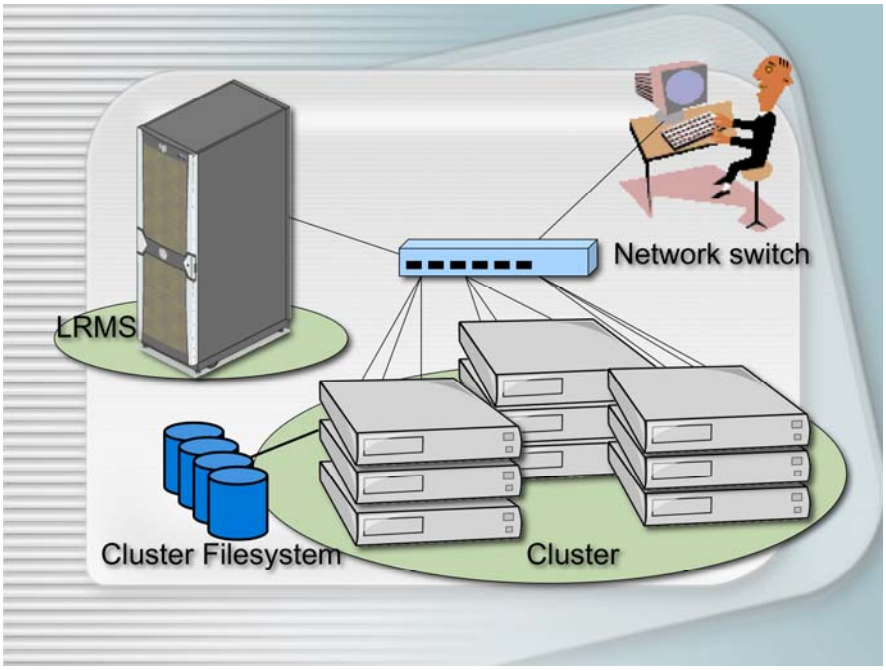


Figure. 2.1 A computing clusters. A cluster filesystem is shared among the nodes of a computer farm. An LRMS head node distributes computing jobs to the nodes of a cluster.

Clusters are often defined as collections of homogeneous servers aggregated for increased performance. Clusters are widely used in the manufacturing domain for things like simulation-based testing and evaluation. The majority of new cluster servers has Gigabit Ethernet interfaces [7], and can range from a handful to literally thousands (in research environments) of servers [28]. As a result, high-density Gigabit Ethernet support is necessary. In addition, low-latency switching is also critical in maintaining application performance across the fabric of a cluster.

Clusters are critical in the evolution of Grid computing. This is because clusters need to be interconnected in order to move to the next phase known as **Intra-Grids** (in analogy with Intranets). Interconnecting separate clusters enables the creation of enterprise and inter-departmental Grids as depicted in Figure 2.2.

The creation of Intra-Grids puts additional constrains on the controlling software layer, or middleware, and the underlying network layer. The middleware must now have a better understanding of resource allocation (computing, storage, network, etc.) because of additional complexity introduced by resources-sharing relationships. Intra-Grids will evolve in a very controlled fashion. For example, two or three clusters may be interconnected between departments within an enterprise to increase processing capacity and share data sets. A good example of this can be found at Magna Steyr in Austria, where clusters of design IBM workstations were interconnected via LSF [20] to increase the computing power dedicated to the batch simulation of the assembly process and the clashes detection [9]. Because the relationship

between clusters is still within one enterprise domain, things like security and authentication, although important, are not as critical as in later phases.

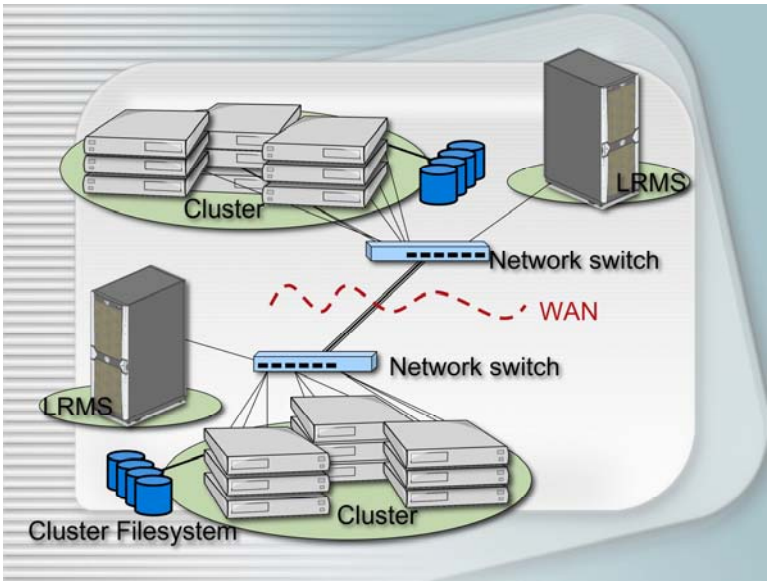


Figure. 2.2 Intra/Extra-Grids: clusters are interconnected in a LAN to form Intra-Grids, or on the WAN to form Extra-Grids.

Extra-Grids are essentially clusters and/or Intra-Grids that are connected between geographically distributed sites within or between enterprise organizations. The two important distinctions here include geographic distribution and inter-enterprise relationships. Now that processing and/or data can be shared between two different organizations, authentication, policy management, and security become critical requirements that the middleware must address. Multi-site load balancing, topology discovery, and application awareness are also important to ensure performance.

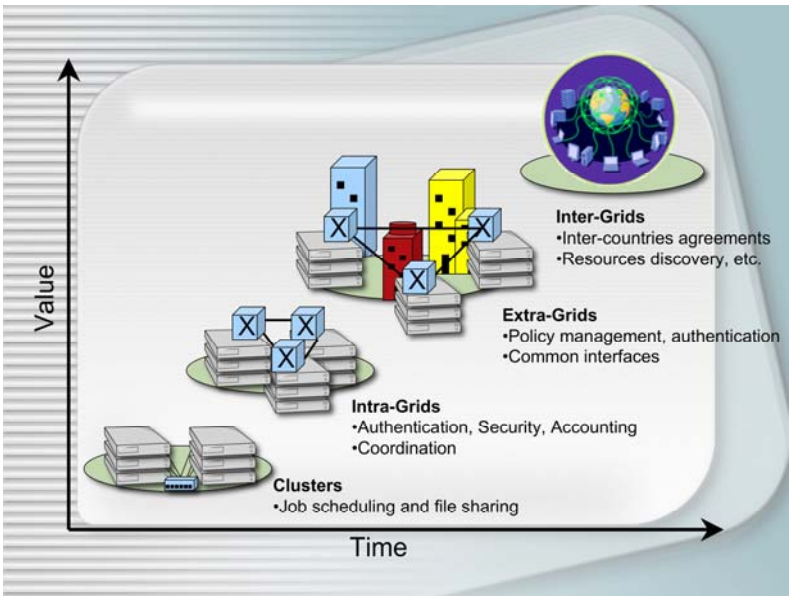


Figure. 2.3 The evolution of the Grid

The final stage in the evolution of Grid computing is the **Inter-Grid** (cf. Figure 2.3). This is the most advanced stage because it embodies two of the primary visions for Grid computing: utility computing infrastructure and Grid services/service providers. Although this is the final stage, and we are not there yet from a commercial perspective, it should be noted that Inter-Grids do exist today in the research and development world [8] [10]. Examples of Inter-Grids (both data and computing Grids) are the WLCG for the High Energy Physics research community, Teragrid [21] in USA for e-Science, EGEE [22] in Europe. An Inter-Grid hosts hundreds or even thousands of users.

2.2 Grid Computing vs. Distributed Computing

Many wonder if Grid computing is not a particular form of distributed computing. However, Grid computing poses many issues that are generally not present when we deal with distributed computing. Here is a non-exhaustive list of differences between Grid and distributed computing.

- Grid computing does not generally focus on one specific type of application but it is supposed to provide a computing infrastructure similar to the one offered by an operating system. Even though the Web is considered to be a predecessor of Grid Computing, it is a good example of distributed computing focusing on information handling.
- Resources are numerous and of many kind.
- Resources are owned and managed by different, potentially mutually distrustful organizations and individuals.
- Resources are potentially faulty and managed with different degree of fault tolerance. A Grid infrastructure is therefore highly dynamic. Resources can appear and disappear while the infrastructure has to guarantee the given quality of service promised to users.
- There are different security requirements.
- Local management policies are different and need to be honored.
- Grid environments are strongly heterogeneous, e.g., they have different CPU architectures, run different operating systems, and have different amounts of memory and disk. Ideally, a running application should be able to migrate from platform to platform.
- Heterogeneous, multi-level networks connect resources. Firewall policies can be different and quite strict from site to site. Resources can also be connected using different technologies.
- Resources are geographically separated (on a campus, in an enterprise, on a continent).
- Sociological and cultural factors influence the success of a Grid computing infrastructure.
- A global name space needs to be enforced in a Grid for data access. More generally, resources need to be described globally with a unified vision.
- One of the most challenging issues in a Grid is that Grid services need to be able to scale to support a very high number of installations.
- Grid infrastructures need to be extensible in order to accommodate smoothly new technologies and new user needs.
- Persistency should be honored by the system. Applications need to read and write data. Such data can be spread around the world and need to be accessed efficiently. Furthermore, many protocols for data access exist and should be supported in order for instance to accommodate the needs of legacy applications.

Complexity management is another issue that spans different sectors: system installation, configuration, support and control, heterogeneity in policies for resource usage, monitoring in large scale, different failure modes, high number of components, etc.

2.3 Grid Architecture

*An **architecture** is a formal description of a system, defining its purpose, functions, externally visible properties, and interfaces. It also includes the description of the system's internal components and their relationships, along with the principles governing its design, operation, and evolution [15][11].*

*A **service** is a software component that can be accessed via a network to provide functionality to a service requester.*

A general, clearly defined Grid architecture that follows the definition above does not exist. This is due to the fact that it is difficult to try to force homogeneity on distributed groups of collaborators. The goal of the CoreGrid project [29] is to provide such a general description of Grid architecture.

A ***Virtual Organization*** (VO) *is a set of individuals and/or institutions defined by specific applications, data and resources sharing rules* [11].

For instance, a virtual organization is the community of Bioinformatics researchers spread worldwide, working on the genome and using a well-defined set of applications (such as BLAST, HMMER, SAM, PFTools for sequence analysis with hidden Markov model approach) and accessing a set of databases storing protein information. VO operation requires that sharing relationships among potential participants can be established. Therefore, it is crucial to have interoperable services. This is achieved with common protocols that define the basic mechanisms by which users and resources negotiate, establish, manage, and exploit sharing relationships in a secure and controlled manner. An effort is under way by the Open Grid Forum (OGF) [30] to organize these protocols under the Open Grid Services Architecture, or **OGSA**, which has grown out of the open standards-based Web Services world. It's called *architecture* because it is mainly about describing and building a well-defined set of interfaces from which systems can be built, all based on open standards like the Web Services Description Language (WSDL) and SOAP (originally called Simple Object Access Protocol, a protocol for exchanging XML-based messages normally using HTTP. Furthermore, OGSA builds on the experience gained from building the Globus Toolkit [14][25], a valuable reference implementation. Designing a viable Grid includes defining multiple layers, each of which plays a critical role. In general, higher layers are more software-centric, whereas lower layers are more hardware centric [7]. The diagram below (cf. Figure 2.4) provides a graphical view of the layers that define Grid computing and the function that each layer provides.

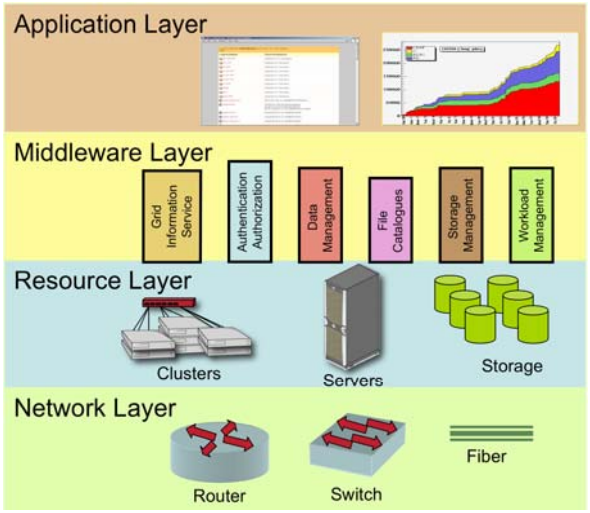


Figure. 2.4 Architecture of a Grid

Application and service-aware software defines the highest layers of the Grid architecture. This includes portals as well as development toolkits. Applications vary from use case to use case in either academia or industry, depending on the problem, as do the portals and development toolkits supporting the applications. Service-awareness provides many management-level functions including billing, accounting, and measurement of usage metrics, all very important topics to track as resources are *virtualized* for sharing among different users, departments, and companies.

The ***Middleware*** layer provides the protocols that enable multiple elements (servers, storage, networks, etc.) to participate in a unified Grid environment. The middleware layer can be thought of as the intelligence that brings the various elements together through software and control. The middleware enables **virtualization** that transforms computing resources into a ubiquitous Grid. There are many different functions and protocols that the middleware layer supports, which are discussed later. As shown in Figure 2.5, the middleware is also organized in layers (similar to the 7 OSI layers in the Internet protocol stack), following the description given in [11]: the ***Middleware Application*** layer provides the interface between the lower middleware layers and the user's applications; the ***Middleware Resource*** and ***Connectivity*** protocols facilitate the sharing of individual resources. Protocols at these layers are designed so that they can be implemented on top of a diverse range of resource types (***Middleware Fabric*** layer). Global services define the ***Collective*** layer, so called because it involves the coordinated ("collective") use of multiple resources.

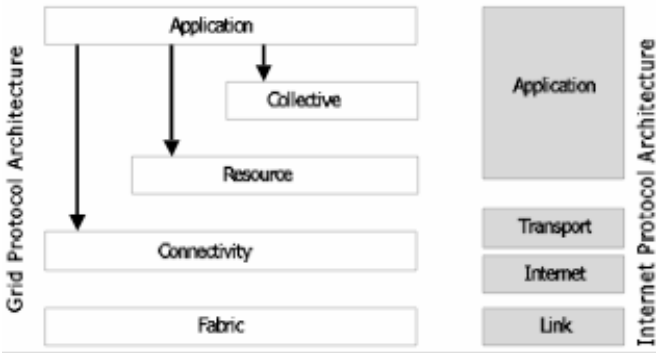


Figure. 2.5 The layered Grid Middleware architecture and its relationship to the Internet protocol architecture. Figure taken from "The Anatomy of the Grid" [11]

The ***Resource*** (or ***Fabric***) layer defines the ***Grid system***. It is made up of the actual resources that are part of the Grid, including primarily servers and storage devices.

The ***Network*** layer is the underlying connectivity for the resources in the Grid.

In what follows we describe the main components of the middleware to build a Grid infrastructure. We will analyze the techniques adopted for authorization and authentication, the information system for resource discovery and monitoring the status of the Grid, the workload management system for computational task management and control, the data management component, and finally storage management. We will refer in particular to the solutions adopted in the WLCG infrastructure.

2.3.1 Security

In order to achieve virtualization of a Grid system, one of the most important issues to be solved is to allow for resource access without imposing complicated and site-specific authentication and authorization procedures.

Authentication *is the process that allows users to prove their own identity.*

Authorization *is the set of operations needed for granting the specific user or service the possibility to perform the requested task.*

Authentication and authorization protocols are part of the Grid middleware Connectivity layer.
The Grid Security Infrastructure (GSI) [13] provides for an implementation of a basic Grid Security Infrastructure. GSI is used for building sophisticated security solutions, mainly for authentication and authorization. It implements the standard (RFC 2078/2743) Generic Security Service Application Program Interface (GSS-API) and is a public-key-based protocol that provides single sign-on authentication, communication protection, and some initial support for restricted delegation.

Single sign-on *allows a user to authenticate once and thus create a proxy credential (a temporary certificate with a limited life time) that an application can use to authenticate with any remote service on the user's behalf.*

Every Grid user needs to have a user certificate, whose private key is protected by a password. Such a certificate is used to generate and sign a temporary certificate, called a proxy certificate (or simply a proxy), which is used for the actual authentication to Grid services and does not need a password. A proxy has, by default, a short lifetime (typically 12 hours) to reduce security risks if it should be stolen (cf. Figure 2.6).

A user needs a valid proxy to interact with the Grid or to submit jobs; those jobs carry their own copies of the user proxy to be able to authenticate with Grid services as they run. For long-running jobs, the job proxy may expire before the job has finished, causing the job to fail. To avoid this, there is a proxy renewal mechanism to keep the job proxy valid for as long as needed. For example, the MyProxy server [57] is the component that provides this functionality (cf. Figure 2.6).

Delegation *allows for the communication with a remote service of user security information that the remote service can use to act on the user's behalf, with various restrictions; this capability is important for chained operations involving multiple services.*

Similar mechanisms can be implemented within the context of other security technologies, such as Kerberos [23][13], although with potentially different characteristics. GSI uses X.509 certificates, a widely employed standard for PKI certificates, as the basis for user authentication. GSI defines an X.509 proxy certificate. GSI uses the Transport Layer Security (TLS) protocol for authentication, although other public key-based authentication protocols could be used with X.509 proxy certificates. The Open Grid Forum drafts define the delegation protocol for remote creation of an X.509 Proxy Certificate and GSS-API extensions that allow this API to be used effectively for Grid programming.

Restricted delegation *allows one entity to delegate just a subset of its total privileges to another entity.*

Such restriction is important to reduce the adverse effects of either intentional or accidental misuse of the delegated credential. Restricted delegation has been shown to be a useful feature and is a critical part of the proposed X.509 Proxy Certificate Profile.

User certificates are *signed* by a Certification Authority (CA) that is trusted by all actors in a Grid environment, i.e. also the remote end of the communication. The remote end (usually at some service provider's site) is able to verify the proxy certificate by descending the certificate signature chain, and thus authenticate the certificate signer. The signer's identity is established by trusting the CA that has signed the certificate. The Certification Authorities participating to a project or enterprise can define a set of rules that each CA has to adopt in order to be trusted within a Grid (cf. Figure 2.6).

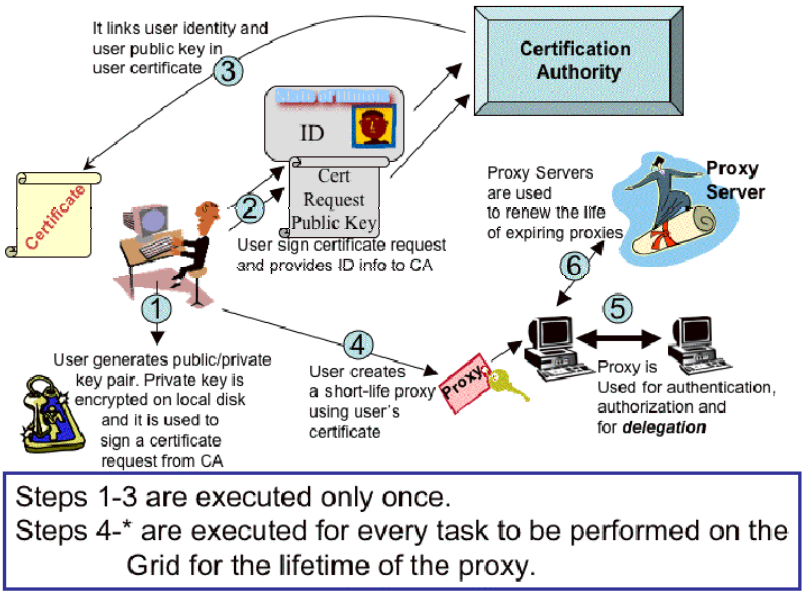


Figure. 2.6 Authentication, authorization, delegation on the Grid.

The last remaining step is user authorization: the requesting user is granted access and mapped to the appropriate resource provider identifiers by checking the proxy (or user) certificate subject (X.509 Distinguished Name, or DN) and properties and authorizing the users. Authorization restrictions can be applied.

Authorization can be managed and enforced by Virtual Organization servers that allow for the description of specific communities with specific privileges and priorities on the Grid resources.
Furthermore, more general security issues and enforcement of local policies must be guaranteed in a Grid environment. Grid systems must have mechanisms that allow users and resource owners to select policies that fit particular needs (restricting access, performance, etc.), as well as meeting local administrative requirements.

Once authentication and authorization have been performed, the user can use Grid resources for the completion of his/her task. In order to guarantee privacy and confidentiality, data can be encrypted so that only authorized users can access them.

Data integrity *guarantees that data has not maliciously or inadvertently been changed during storage or transmission without proper authorization.*

Data confidentiality *forbids an unauthorized party to access a particular piece of data, again either in transit or in storage.*

2.3.2 The Information System

Other basic protocols (***Resource layer***) of a Grid infrastructure are those that allow for the implementation of an Information System.

*A **Grid Information Service** allows for the registration of available resources and services in the Grid in order to provide for discovery and monitoring features.*

Access to information on Grid resources is a necessity in order to perform common tasks such as resource discovery, matching job requirements with available resources, accessing files or presenting monitoring information. Both middleware services (such as workload and data management) and applications (such as monitoring tools etc.) require an interface to the Grid Information Service (IS) to find out about resources as well as status information.

Hierarchical directory services are widely used for this kind of applications, due to their well defined APIs and protocols. A first prototypal implementation based on the Lightweight Directory Access Protocol (LDAP) can be found in the Globus Toolkit version 1 and 2.

Overall, the main issue with using existing directory services based on LDAP is that they are not designed to store dynamic information such as the status of computing (or networking) resources. Therefore, the OGF has created a Grid Monitoring Architecture (GMA) [58] that defines general Grid information and monitoring systems. In general, a Grid Information Service should provide a number of useful features, such as:

- A well-defined data model and a standard and consolidated way to describe data.
- A standardized API (Application Programming Interface) to access data on the directory servers.
- A distributed topological model that allows for data distribution and delegation of access policies among institutions.

One important point to make a Grid Information Service effective is to define a complete and extensible ***schema*** that fully describes Grid resources and services and their properties. The Grid Laboratory for a Uniform Environment (GLUE) joint effort between Grid projects in Europe and the USA together with an OGF working group [27] are working on the standardization of a schema. The common schema describes computing, data and storage resources and services in a homogeneous way and allows for interoperability between different Grid infrastructures in the world [8][37].

2.3.3 The Workload Management System

*The **Workload Management System** is one of the **Collective** services in the Grid architecture. It allows participants to request the allocation of one or more resources for a specific purpose and the scheduling of tasks on the appropriate resources. It also supervises and manages the workflow between multiple service components of the Grid in order to successfully complete the given user task.*

One of the most important components of a workload management system is the protocol that regulates the access to Grid computing resources. Such protocol is responsible for operating a set of computing resources under site-specific allocation policies; this is often done by a local resource management system/scheduler (such as LSF, PBS, and Condor etc. [20][59][56]). In particular, the Grid service that allows for access to the nodes of a local computing farm is responsible for:

- Processing resource requests, by either creating the process satisfying the request, or denying that request;
- Enabling remote job monitoring and management;
- Periodically updating the information about the current status and characteristics of the resources it manages.

A ***Computing Elements*** (***CE***) *is the service that acts as a gateway to local nodes of a cluster or supercomputer. A job dispatched to a Computing Element is then passed to a local scheduler for its execution on the controlled farm nodes.*

Worker Nodes (***WN***) *belong to a computing farm or cluster attached to the various CEs. They represent the processing power of the Grid.* (cf. Figure 2.7)

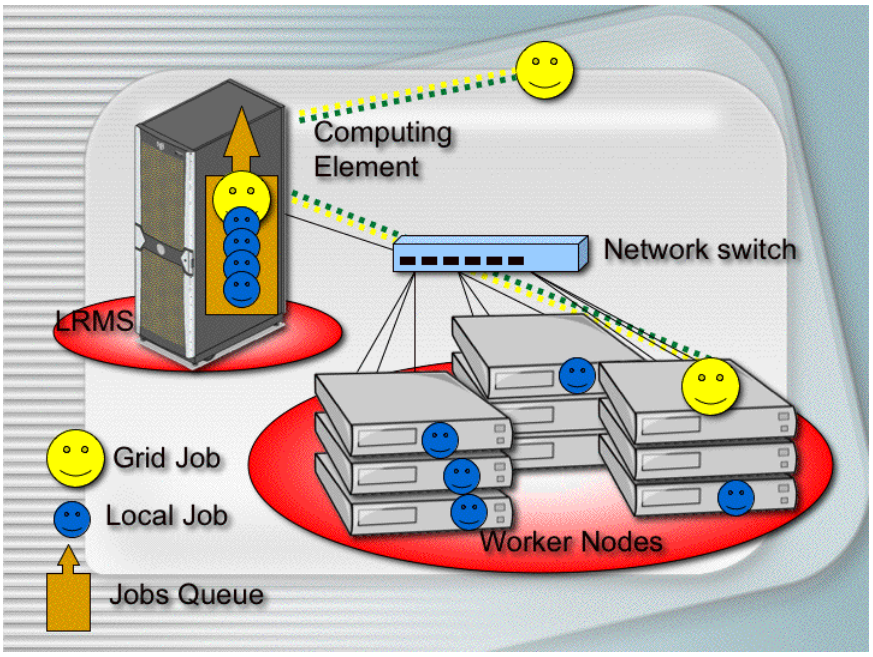


Figure. 2.7 Computing Element and Worker Nodes: A Grid job lands on a Grid Computing Element (CE) where a Local Resource Manager System (LRMS) queues it together with other local jobs to be executed on Worker Nodes (WN) of a computing farm.

The Grid global scheduler [10] is one of the most critical components of the resource management systems, since it has the responsibility of assigning resources to jobs in such a way that the application requirements are met, and of ensuring that the resource usage limits granted to the user are not exceeded. It interacts with other Grid services such as the Information System, Grid file catalogs, and storage services in order to correctly schedule a job to the most appropriate resources and possibly prepare the execution environment making available the necessary input data or reserving space for the output.
Existing Grid schedulers can be classified according to three factors, namely their organization (that may be centralized, hierarchical, or distributed), their scheduling policy (that may optimize either system or application performance), and the state estimation technique they use to construct predictive models of application performance.

The ***centralized organization***, as it is in Condor, is under the control of a single entity, which receives and processes all the allocation requests coming from Grid users. However, while this approach has the advantage of providing the scheduler with a global system-wide view of the status of submitted jobs and available resources, so that optimal scheduling decisions are possible, it is poorly scalable and non tolerant

to failures. A centralized scheduler represents a performance bottleneck (i.e., it can be overloaded with many jobs) and a single point of failure.

In a ***distributed organization***, like in AppLeS [31], Ninf [32], NetSolve [33] and AliEn[34], the scheduling decisions are delegated to individual applications and resources. In particular, each application is free to choose (according to suitable policies) the set of resources that better fit its needs, and each resource is free to decide the schedule of the applications submitted to it. In this approach there are no bottlenecks and single points of failure but, being the scheduling decisions based on local knowledge only, resource allocation is in general sub-optimal. In the case of AliEn, job agents running on a given resource check the local environment and if ok they pull the next job requiring those resources from a central queue. This approach is very efficient but the single task queue may represent a single point of failure, even if quite manageable.

Finally in the ***hierarchical approach***, as adopted in Darwin [35] and Nimrod/G [36], the scheduling responsibilities are distributed across a hierarchy of schedulers. Schedulers belonging to the higher levels of the hierarchy make scheduling decisions concerning larger sets of resources (e.g., the resources in a given continent), while lower-level schedulers are responsible for smaller resource ensembles (e.g., the resources in a given state). At the bottom of the hierarchy there are schedulers that schedule individual resources. The hierarchical approach tries to overcome the drawbacks of the centralized and the distributed approach, while keeping their advantages at the same time.

The other important feature of a grid scheduler is the adopted ***scheduling policy***. The schedulers of Condor [56] and Darwin adopt a *system-oriented* policy, aimed at optimizing system performance and resources utilization. However, they do not deal with resource co-allocation and advance reservation. Systems like AppLes, NetSolve, Nimrod/G, and Ninf adopt *application-oriented* scheduling policies. In HEP there is also the need of scheduling techniques able to maximize *user application performance*.

In order to obtain satisfactory performance, a scheduler must employ ***predictive models*** to evaluate the performance of the application or of the system, and use this information to determine the allocation that results in best performance. Condor, Darwin, Nimrod/G, and Ninf adopt *non-predictive* schedulers. However, assuming that the current resource status will not change during the execution of applications may result in performance much worse than expected because of the possible presence of contention effects on the resources chosen for the execution of an application. AppLeS, and NetSolve address this problem by adopting *predictive* schedulers. Predictive techniques usually require a higher computational cost than their non-predictive counterparts.

2.3.4 Data Management and Replication

In a Data Grid infrastructure transparent and efficient access to data is vital. In the HEP environment physicists all around the world compete to publish the results obtained by the analysis of the data produced by some particle detector present at some location. The physical location of the detector and the data acquired should not penalize researchers sitting in a different lab.

A ***replica*** *is a managed copy of a file.*

Data replication is a well-known and accepted technique for optimizing data access and providing fault tolerance. This is achieved by storing multiple copies of data at several locations. The topology and the latency of the network have an important influence on the replication strategy to be used. Therefore,

Grid data management *focuses on file replication services and its main objectives include optimized data access, caching, file replication and file migration [38][39].*

The most important functionalities of Grid Data Management (DM) services and protocols are:

- Management of a ***universal namespace*** for files (using replica catalogs)
- Secure, reliable and efficient data transfer between sites
- Synchronization and consistency of remote copies
- (Optimized) wide-area data access/caching
- Management of meta-data like indices and file meta-data
- Interface to heterogeneous mass storage systems

In a Grid environment, ***files*** can have ***replicas*** at many different sites. Ideally, the users do not need to know where a file is located, as they use logical names for the files that the Data Management services will use to locate and access them. In order to guarantee a unique namespace, several different ***file name conventions*** are known: ***Grid Unique IDentifier*** (GUID), ***Logical File Name*** (LFN), ***Storage URL*** (SURL) and ***Transport URL*** (TURL). While the GUIDs and LFNs identify a file irrespective of its location, the SURLs and TURLs contain information about where a physical replica is located, and how it can be accessed (cf. Figure 2.8).

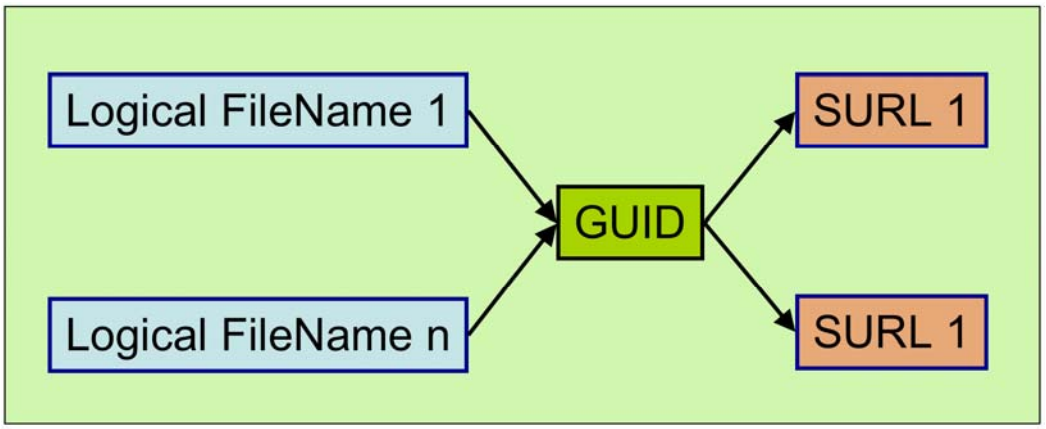


Figure. 2.8 Grid filenames

A file can be unambigously identified by its ***GUID***; this is assigned the first time the file is registered in the Grid, and is *based on the UUID standard to guarantee its uniqueness.* In order to locate a file in the Grid, a user will normally use an LFN.

LFNs *are usually intuitive, human-readable strings, and they are chosen by the user.* A Grid file can have many LFNs, in the same way as a file in a Unix file system can have many links.

The ***SURL*** *provides informations about the physical location of the file since, for instance, the hostname of the server hosting a replica is coded in the SURL itself.*

Finally, *the* ***TURL*** *gives the necessary information to retrieve a physical replica, including hostname, path, protocol and port (as for any conventional URL), so that the application can open or copy it.*
There is no guarantee that the path, or even the hostname, in the SURL is the same as in the TURL for the same file. For a given file there can be as many TURLs as there are data access protocols supported by the SE. Figure 2.9 shows the relationship between the different names of a file. The mappings between

LFNs, GUIDs and SURLs are kept in a service called a **File Catalog**, while the files hemselves are stored in Storage Elements.

The building blocks of the Data Management architecture are: Replica Managers, Replica and Metadata Catalogs, File Copier and the Consistency Services [38][39][40][41] (cf. Figure 2.9). The **Replica Manager** manages file transfers (replication) by using the **File Copier** service and the replica catalog information through the replica catalog service. One of the first implementations of a Replica Manager has been provided by the EDG project: GDMP (Grid Data Mirroring Package) [42][43] is a file replication software package that implements most of the replica manager functionalities. The main tasks of the replica manager are to securely, consistently and efficiently copy files between Grid sites and update the Grid Replica Catalogues when the copy process has successfully terminated. Using some performance information such as the current network throughput and latency and the load on the storage services at a site it is possible to optimize the access to the replicated files. The **Replica Optimizer**'s duty is to select the "best" replicas. The File Copier (also called Data Mover) is an efficient and secure file transfer service that has to be available at each site. The **Consistency Service** [41] [44] has to guarantee the synchronization of the modifiable file replicas when an update occurs.

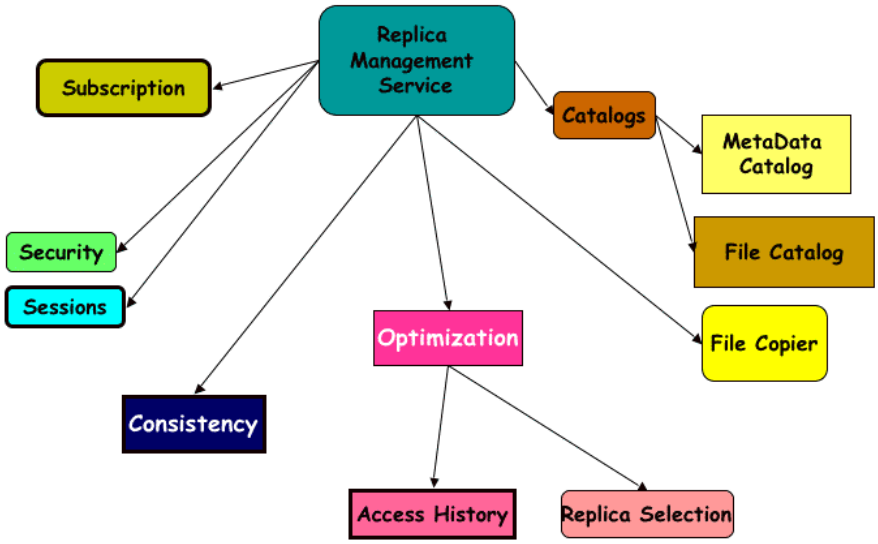


Figure. 2.9 Data Management Service components

Data Management has to deal as well with the heterogeneity of storage systems and thus ***Storage Elements*** (***SE***) defined later. The interface to an SE has to be unique regardless of the underlying storage technology.

2.3.5 Storage Management and Access

*A **Grid Storage Element (SE)** is the set of hardware and software solutions adopted in order to realize a storage Grid service. It allows users to consistently and securely store files at a Grid site.*

An SE is a basic storage resource, which provides a uniform interface to storage in a Data Grid. In terms of hardware, a Storage Element consists of a large disk pool or Mass Storage System (MSS) that uses robotic tape libraries.

Persistent storage *is a storage space dedicated to long lasting data.* Persistent storage systems are, for instance, tape-based systems.

Volatile storage *is a storage space where data can be stored for a limited period of time, as long as the data is accessed or as long as there is enough space available to satisfy further incoming requests.*

The current storage system implementations include MSS such as HPSS and Castor as well as distributed file systems like AFS or NFS. They will be described in detail later in this thesis. All these implementations can potentially co-exist and the Storage Element subsystem has to provide for a generic standard interface hiding the specific access mechanisms (and their complexity) foreseen by the local storage implementation.

From a software perspective, besides file access, control and transfer protocols, other functionalities such as those listed below provide for the basic building blocks for a storage service:

- Mechanisms for putting (writing) and getting (reading) files
- Mechanisms for file "pinning" (i.e. stopping deletion from on-line storage of currently accessed files).
- Third party and high-performance (e.g., striped) transfers
- Mechanisms for reading and writing subsets of a file
- Mechanisms for executing remote data selection or reduction functions
- Management mechanisms that allow control over the resources allocated to data transfers (space, disk bandwidth, network bandwidth, CPU)
- Advance reservation mechanisms
- Enquiry functions for determining hardware and software characteristics as well as relevant load information such as available space and bandwidth utilization.

Several of the above mentioned building blocks are defined by the ***Storage Resource Manager (SRM)*** specification [60] that we will discuss later in this thesis in more detail.

2.4 Current Middleware Solutions

Multilayer architectures such as Grids demand strong standardization, because equipment from different suppliers will be part of distributed deployments. There are several grid-specific standard initiatives and several projects that implemented a Grid infrastructure in order to study the possible issues and propose input to the standardization discussions. Among those the most popular are the ones described in the following paragraphs. We categorize the middleware solutions according to the architectural components listed in the previous subsection.

2.4.1 Condor

The Condor project started to be active the beginning of 1980. It is one of the real predecessors of Grid computing since it provides users worldwide with transparent access to heterogeneous computing resources that are otherwise idle. At the University of Wisconsin in Madison Condor manages more than

1000 workstations. Condor is quite robust and has been used by companies and in educational and research environments to build effective computing infrastructure. Many Condor components (Condor-G, ClassAds, etc.) are being used by projects such as WLCG and EGEE-II to build today's Grid infrastructures.

Like other workload management systems, Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Furthermore Condor provides check-pointing mechanisms to migrate jobs to a different machine when the current machine becomes busy or unavailable and capabilities to redirect all the job's I/O requests back to the submit machine.

Through the *Condor-G* mechanism, Condor can also manage resources part of a Globus Grid infrastructure, submitting jobs to them.

Security
Condor relies on the underlying Unix UID/GID authentication/authorization method. Jobs in the Condor managed pool of machines can run either under the UID/GID of the user who submitted the job or as user nobody. If the major Condor services are not run as root, the functionality of the entire system is very much reduced, besides introducing security issues.

If the Andrew File System (AFS) or a Kerberos based filesystem is used at a site, then Condor will not guarantee data access. In fact at the moment (version 6.8.1) Condor cannot authenticate itself to AFS or to a Kerberos based filesystem.

Information System
Condor provides an internal Information System with its own proprietary schema. The *ClassAd* mechanism offers an extremely flexible and expressive framework for matching resource requests (jobs) with resource offers (machines). Users can easily describe jobs requirements. Likewise, machines can specify requirements and preferences about the jobs they can run. These requirements and preferences can be described in expressions that are evaluated to enforce the desired policy. Condor simplifies job submission by acting as a matchmaker of ClassAds continuously reading all job ClassAds and all the machines ClassAds, matching and ranking job ads with machine ads.

Workload Management
Beside the job matching facility, Condor offers as well *meta-scheduling* capabilities. Through a Direct Acyclic Graph (DAG), users can express job dependencies, where the output produced by one job is the input for the next task. The jobs are nodes (vertices) in the graph, and the edges (arcs) identify the dependencies. Condor finds machines for the execution of programs, but it does not schedule programs (jobs) based on dependencies. The Directed Acyclic Graph Manager (DAGMan) is a meta-scheduler for Condor jobs. DAGMan sorts out the order in which jobs described by a DAG should be executed, submits the jobs to Condor and processes the results. Nodes in the graph can also be data placement jobs, to move data necessary for computation in the correct place. Data placement jobs are managed by the ***Stork*** subsystem.

Data Management
Stork is a scheduler for data placement jobs. Data placement jobs are treated as computational jobs: they are queued, managed, queried and autonomously restarted upon error. Through the use of different modules Stork can move data using different protocols: ftp, http, gsiftp, srb, srm, unitree, etc. They will be described in more detail in the next chapters. ClassAds has been extended to introduce new keywords that are useful to describe data management operations.
Storage Access and Management

In terms of storage management the Condor project has recently proposed the **NeST** product, a user-level software-only storage appliance.

*An **appliance** is any tool that performs one function well.*

NeST provides a virtual protocol layer to allow for different authentication as well as data transfer protocols. An abstract storage interface is used to optimally utilize a wide range of physical storage devices. NeST supports functions such as space reservation, service level, local policies enforcement and advertisement of functionality and status information. NeST will be described later in more details.
The Condor software products are quite robust and production ready, however they present a few problems:

- The systems tend to be monolithic and subject to non-open source license.
- Support for data transfer of big amounts of data (order of Petabyte) is quite rudimentary.
- The internal system communication is done using insecure network channels.

2.4.2 Globus

The Globus Project started in 1996 under the initiative of scientists at Argonne National Laboratory (led by Ian Foster and Steve Tuecke) and the University of Southern California's Information Sciences Institute (led by Carl Kesselman). The project focused on how to help collaborative science applications make better use of the Internet infrastructure for large data requirements. The Globus Toolkit is a set of open source building blocks developed by the Globus Alliance to prove the feasibility of Grid infrastructures and take advantage of the cutting edge of hardware interoperability and inter-organizational data sharing capabilities. Even if at the prototypal level, the Globus Toolkit has been used in many successful Grid implementations in e-science, such as EDG, WLCG, OSG and NDGF. It has also become the de facto building block for Grid in the enterprise (SAP, etc.), and a key part of many enterprise vendors' strategies (IBM, HP, Sun). The Globus Consortium is a non-profit organization dedicated to the continued commercial advancement of the Globus Toolkit, and has the support of leading enterprise hardware and software vendors, the original Globus Team, and the open source Grid development community.

The first release 1.0.0 of the Globus Toolkit (GT) appeared at the end of 1998 but it was only with the subsequent versions 1.1.3 and 1.1.4 released in 2000 that the first application reports were given [25][45], outlining many deficiencies of the software even if the toolkit provided was really establishing a good base for a Grid infrastructure and a first architectural view.

With the involvement of industry and the release of GT3 in 2003, Globus promotes the new OGSA that represents an evolution towards a Grid system architecture based on Web services concepts and technologies. This was the start that brought the Globus Alliance (an international collaboration for the research and development of Grid technologies) to the definition of the Web Service Resource Framework (WSRF) announced with IBM and HP on January 20, 2004.

By default, standard Web services are stateless, which sometimes is considered as a restriction for building Grid Services. Therefore, WSRF introduces state to a Web service by extending standard Web service technologies such as WSDL and SOAP. It additionally defines the conventions for managing the lifecycle of services, the discovery, inspection, and interaction with stateful resources in standard and interoperable ways.

The aim of the GT was to lay out a possible Grid architecture based on services and to identify the areas for standardization and the need of common protocols. Among others, the Globus Toolkit provides tools and libraries for all architectural areas discussed in the previous subsection:

- Security
- Information System
- Resource Management
- Storage Access and Data Management

Security
Security in the Globus Toolkit is based on the **GSI** protocol built on an extension of the TLS protocol, as explained earlier. The Globus Toolkit provides a command line interface and API to GSI in order to validate and cache user credentials. It does not enforce any user policies or roles. Local policies can be enforced via a Generic Authorization and Access (GAA) control interface. The service relies on the OpenSSL library to implement an X.509 based security infrastructure.

The **Community Authorization Server** (CAS) [61] allows a virtual organization to express policies regarding resources distributed across a number of sites. A CAS server issues assertions to users, granting them fine-grained access rights to resources. Servers recognize and enforce the assertions. CAS is designed to be extensible to multiple services and is currently (GT 4.0) supported by the GridFTP server described later.

Information System
The Monitoring and Discovery Service (MDS) provides a wrapper to LDAP for querying Grid resources (see above). It is a hierarchical model where resources publish their information via the Grid Resource Information Service (GRIS). Computing and storage resources at a site run a piece of software called an Information Provider, which generates the relevant information about the resource (both static, like the type of Storage Element, and dynamic, like the used space in an SE). This information is published via the GRIS running on the resource itself.

A Grid Resource Information Protocol (GRIP, currently based on the LDAP protocol) is used to define a standard resource information protocol and associated information model or schema. The GRIS pushes information to the Grid Information Index Server (GIIS). A soft-state resource registration protocol, the Grid Resource Registration Protocol (GRRP), is used to register resources with GIISes. The GIIS caches the information and invalidates it if such information is not refreshed in a given configurable timeout interval. Through this mechanism it is possible to build a hierarchical system to retrieve global information. A data schema defines the nature and the structure of the data being published.

The MDS implements the GLUE Schema using OpenLDAP, an open source implementation of the LDAP. Access to MDS data is insecure, both for reading (clients and users) and for writing (services publishing information). The LDAP information model is based on entries (objects like a person, a computer, a server, etc.), each with one or more attributes. Each entry has a Distinguished Name (DN) that uniquely identifies it, and each attribute has a type and one or more values. Entries can be arranged into a hierarchical tree-like structure, called a Directory Information Tree (DIT). Figure 2.10 schematically depicts the Directory Information Tree (DIT) of a site: the root entry identifies the site, and entries for site information, CEs, SEs and the network are defined in the second level.

The LDAP schema describes the information that can be stored in each entry of the DIT and defines object classes, which are collections of mandatory and optional attribute names and value types. While a

directory entry describes some object, an object class can be seen as a general description of an object, as opposed to the description of a particular instance.

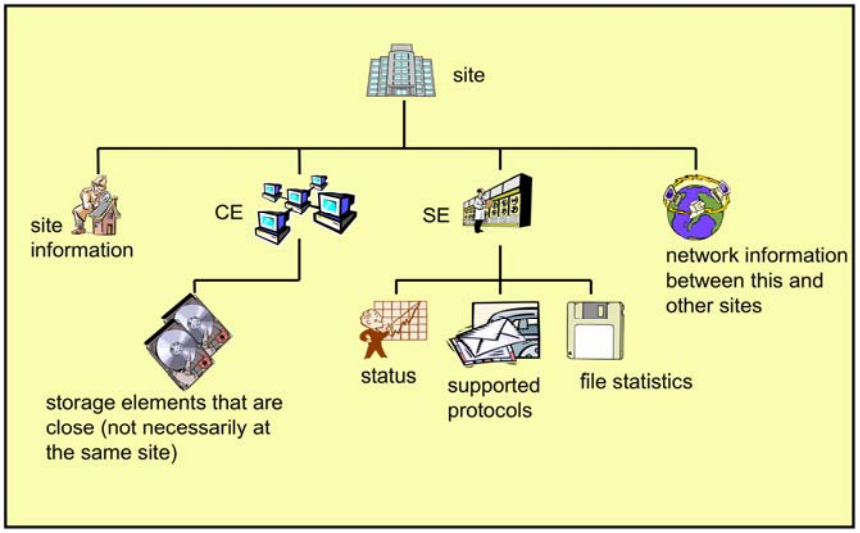


Figure. 2.10 The Directory Information Tree (DIT)

Resource Management
Management protocols are used to negotiate access to shared resources, specifying, for example, resource requirements such as advanced reservation and quality of service, as well as operations to be performed, such as process creation, or data access.

The Grid Resource Allocation and Management (GRAM) protocol is an HTTP-based protocol used for allocation of computational resources and for monitoring and control of computation on those resources. Based on the GRAM protocol, the GRAM service is composed of three sub-parts: a gatekeeper, a job manager and a reporter. The GRAM service architecture is shown in Figure 2.11. A Web Services (WS) based GRAM is distributed with GT3 and GT4. In particular, it is a WS version of the original GRAM described here.

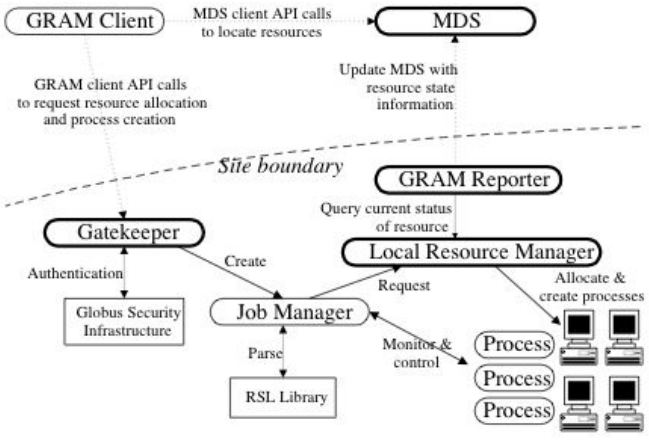


Figure. 2.11 The GRAM architecture

Via the Resource Specification Language (RSL) the user can specify resource requirements interpreted by the GRAM client. The GRAM client library is used by the application: it interacts with the GRAM

gatekeeper at a remote site to perform mutual authentication and transfer the request. The gatekeeper responds to the request by performing mutual authentication of user and resource (using the GSI service), determining the local user name for the remote user, and starting a job manager, which is executed as the selected local user and actually handles the request. The job manager is responsible for creating the actual processes requested. This task typically involves submitting a request to an underlying resource management system (Condor, LSF, PBS, NQE, etc.), although if no such system exists, the fork system call may be used. The job manager is also responsible for monitoring their state, notifying a callback function at any state transition. The job manager terminates once the jobs for which it is responsible have terminated. The GRAM reporter is responsible for storing in MDS (GRIS) various information about the status and the characteristics of resources and jobs.

Storage Access and Data Management
While in other areas the Globus Alliance has contributed strongly to create the building blocks for a Grid infrastructure, in terms of Storage Access and Management the Globus Toolkit lacks a concrete proposal. Among the main contributions in the area of Storage Access and Data Management we can list the following protocols and services:

- the GridFTP protocol
- the Global Access to Secondary Storage (GASS) service
- the Replica Location Service
- the Reliable File Transfer Service
- the Data Replication Service Manager

GridFTP is the only protocol implemented in the Globus Toolkit concerning data access/transfer on a storage system. GridFTP is an extension of the FTP protocol based on RFC949, RFC2228 and RFC2389. Among the main features we can list the following:

- A high throughput, reliable, secure and robust data transfer mechanism.
- GSSAPI security (PKI and Kerberos) support.
- Automatic negotiation of TPC buffer/window sizes.
- Parallel data transfer.
- Third-party control of data transfer.
- Partial file transfer.

The ***Global Access to Secondary Storage (GASS)*** service is the Globus tool that simplifies the porting to and running of applications that use file I/O in the Globus environment. It consists of libraries and utilities that eliminate the need for manually transferring files from/to remote sites and/or share disks by means of distributed file systems. In order to use the Globus GASS libraries, applications need to be modified to call the appropriate functions for file access. GASS supports various protocols for file transfer: x-gass, ftp and http. The GASS architecture is shown in Figure 2.12.

The GRAM uses the GASS service to copy the input files and the executable from the submitting machine to a disk cache in the executing machine by means of one of the 3 servers shown in the picture (cf. Figure 2.12). When the job is completed, the output and error files are copied back to the submitting machine using the same mechanism. Some APIs are also available to perform the management of the remote cache. This service is only present in GT2 and GT3.

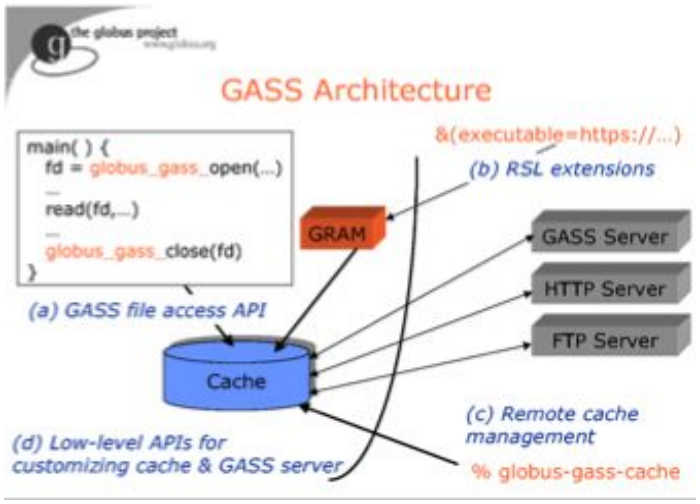


Figure. 2.12 The GASS architecture (source Globus)

The ***Replica Location Service (RLS)*** is a service developed in cooperation by the Globus and EDG data management teams. It is an attempt to provide an answer to the need of a unique file name space in the Grid. For optimization reasons, files can be replicated (copied) in a Grid infrastructure and have many physical instances.

An application needing to access one instance of a file needs to transparently access such a file, independently of its location and storage properties. The RLS service provides a mechanism for storing/associating to a *logical file name* a *Grid Universal Identifier (GUID)*, and to such identifier the set of physical file handles at the various Grid locations. Using the RLS, a Replica Manager or Data Access Optimization Service can return the "best" available replica handle to an application asking to access a given file via its logical file name.
The ***Reliable File Transfer Service (RFTS)*** is a Web service that provides interfaces for controlling and monitoring third party file transfers using GridFTP servers.

The ***Data Replication Service (DRS)*** allows applications to securely control where and when copies of files are created, and provides information about where copies are located in the Grid. The DRS conforms to the WS-RF specification and uses the Globus RLS to locate and register replicas and the Globus RFT to transfer files. Files are assumed to be mostly read-only. No consistency of replicas is guaranteed: it is possible for copies to get out of date with respect to one another, if a user chooses to modify a copy. The DRS can also handle collections of files.

2.4.3 Legion and Avaki

Legion is a Grid Project started in 1993 at the University of Virginia. At that time the word "Grid" was not used but instead the term "*metasystem*" was popular. Given the dramatic changes in wide-area network bandwidth in addition to the expected faster processors, more available memory, more disk space, etc. the researchers participating to the Legion project intended to construct and deploy large-scale "metasystems" for scientific computing, with a design goal to build a general purpose meta-operating system and include a phase of technology transfer to industry.

Legion is a Grid architecture as well as an operational infrastructure. The NPACI-Net created in 1998 was the first infrastructure using the Legion middleware and consisting of hosts at UVa, Caltech, UC Berkeley, IU, NCSA, the University of Michigan, Georgia Tech, Tokyo Institute of Technology and the Vrije Universiteit, Amsterdam. NPACI-Net continues still today with additional sites such as the University of Minnesota, SUNY Binghamton and PSC. Supported platforms include MS Windows, the Compaq iPaq, the T3E and T90, IBM SP-3, Solaris, Irix, HPUX, Linux, True 64 Unix and others.

In 1999, Applied MetaComputing was founded to carry out the technology transition. In 2001, Applied MetaComputing became Avaki and acquired legal rights to Legion from the University of Virginia. Avaki has collected success in the business environment. Only recently, in September 2005, Sybase has acquired Avaki but the main line of the software has not changed, even if it has been hardened, trimmed-down and focused on commercial requirements.

Legion is an object-based system composed of independent objects that communicate with one another by method invocation. Because of that, Legion has a peculiar and very extensible architecture. Everything in the system is an object: files, applications, application instances, users, groups, etc. Users can write their own class objects to determine and even change the system-level mechanisms that support their objects. Legion contains default implementations of several useful types of classes.

All Legion objects have a name, state (which may or may not persist), meta-data associated with their state and an interface. At the heart of Legion is a ***global naming scheme***. Security is built into the core of the system.

Legion ***core objects*** implement common services such as naming and binding, and object creation, activation, deactivation, and deletion. Core Legion objects provide the mechanisms that classes use to implement policies appropriate for their instances.
Class objects are responsible for many operations that are usually considered to be at the system level. This is one of the great advantages of the Legion model. Also, system-level policies are encapsulated in extensible, replaceable class objects, supported by a set of primitive operations exported by the Legion core objects.

Objects have *attributes* that describe the object characteristics and their status. Such attributes are stored as part of the state of the object, and can be dynamically retrieved or modified only by invoking object operations. Attributes (object metadata) are stored into meta-data databases called *collections*. Collections are the equivalent of the Globus ***Information Systems*** and the attributes define possible ***information schemas***.

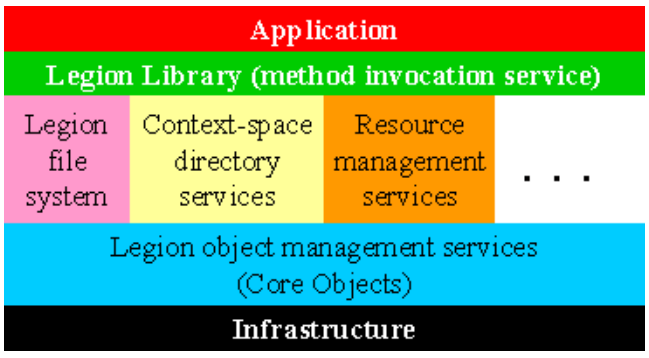


Figure. 2.13 The Legion architecture

Security
Legion provides mechanisms for ***authentication, authorization and data integrity/confidentiality***.

Users authenticate in Legion using a username and a password. The system creates an authentication object for each user. The authentication object contains the user's private key, her encrypted password and user profile information as part of its persistent state. The login process generates a proxy authentication object. The information (password) provided by the user at login time is compared with the one stored in the state of the authentication object. Only a match guarantees access to the Grid. The state of the authentication object is stored on disk and protected by the local operating system. Legion does not enforce any specific authorization system: Kerberos, LDAP, RSA, etc. are all allowed systems. After authentication, the authentication object generates and signs an encrypted (non-X.509) credential that is passed back to the caller. The credential is stored and protected by the security of the underlying operating system. Credentials are propagated during object invocations and are used for further access to Grid resources, just as in the case of Globus.

In terms of authorization Legion is also very flexible. By default, Access Control Lists (ACLs) are available with "allow/deny" lists specifying who is allowed or not allowed to execute a specific action, access certain data, etc. ACLs can be extended to a great variety of Legion objects to allow for the implementation of peculiar application-specific policies.

Legion also allows for data integrity and confidentiality. Legion allows for three modes of operations for data access: private, protected and none. In the private mode, all data is fully encrypted. In the protected mode, the sender computes a checksum (e.g., MD5) of the actual message. In the third mode, "none", all data is transmitted in clear text except for credentials. Legion uses OpenSSL for encryption.

Data Access
For storage on a machine, Legion relies largely on the underlying operating system. Persistent data is stored on local disks with the appropriate permissions such that only authorized users can access them.

Data access is possible in Legion via three mechanisms: by the Data Access Point (DAP) Legion-aware NFS server, the command line utilities similar to the ones provided by a Unix file system (ls, cat, etc.) or by the Legion I/O libraries similar to the C stdio libraries. The DAP responds to NFS protocols and interacts with the Legion system. When an NFS client on a host mounts a DAP it maps the Legion global name space into the local host file system and provides access to data throughout the Grid without the need of installing Legion software. The DAP supports the Legion security mechanisms: access control is with signed credentials, and interactions with the data Grid can be encrypted.

Legion does not offer solutions to make efficient use of robotic libraries, disk pools, high-performing parallel file systems or hierarchical storage managers (described later). Therefore a complete Grid solution for storage management and access is lacking in Legion.

Resource Management
Application scheduling is conceived in Legion keeping in mind two fundamental requirements: sites are totally autonomous and they can establish local policies, as they wish; users must also be free to setup their own scheduling decisions in order to achieve maximum performance. The Legion scheduling module consists of three major components: a resource state information database (the Collection), a module which computes request (object) mapping to computing and storage resources (the Scheduler), and an activation agent responsible for implementing the computed schedule (the Enactor). They work as their equivalents in Globus. The Collection queries the resources to find out about their characteristics and status. The Scheduler queries the Collection to find out which resources matches the requirements

for the application to be run. The Enactor places objects on the host and monitors them. The structure of the Legion middleware is shown in Figure 2.13.

The Legion system is fully integrated and interacts with industrial mechanisms for the distribution of object-oriented systems. It substitutes operating system services and therefore can pose quite some restrictions on the applications that use it.

2.4.4 gLite

The WLCG infrastructure (described in Section 2.5.1) uses a Grid middleware system called gLite [49] that has been developed by the EGEE project. In what follows we describe in details the main components of the middleware. This is needed in order to introduce fundamental concepts used through the rest of this work. A detailed description of the WLCG middleware functionality can be found in [51]. The gLite middleware provides an integrated set of services that allow for secure job execution and data management in a distributed Grid environment. Figure 2.15 shows the main components of the gLite middleware. They will be explained in details in this section.

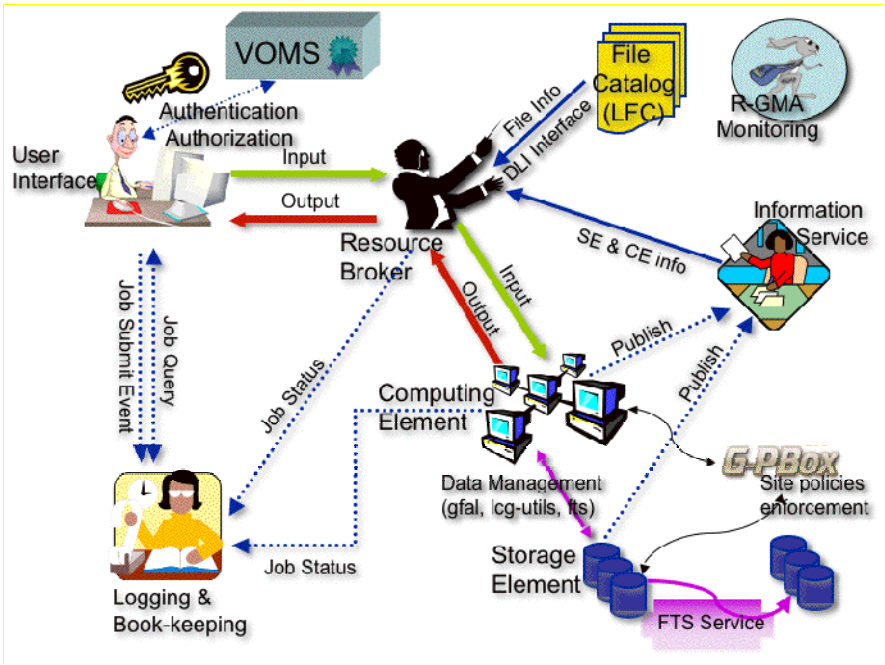


Figure. 2.14 The main components of the gLite middleware

The access point to the gLite middleware is the so-called ***User Interface (UI)*** that provides for command line tools and APIs for secure job submission, data management as well as access to the information system. Typically, the UI can be regarded as a set of client tools installed on a single machine where registered users have their personal accounts. In detail, the following Grid operations are provided:

1. Information management:
   - Retrieve the status of different resources from the Information System;

2. Job management:
   - list all the resources suitable to execute a given job;

- submit jobs for execution;
- cancel jobs;
- retrieve the output of finished jobs;
- show the status of submitted jobs;
- retrieve the logging and bookkeeping information of jobs;

3. Data management:
   - copy, replicate and delete files from the Grid;

Security
gLite uses GSI (i.e. X.509 certificates) for basic user *authentication*. The *authorization* of a user on a specific Grid resource can be done in two different ways:

1. The first is simpler, and relies on the grid-mapfile mechanism. The Grid resource has a local grid-mapfile, which maps user certificates to local accounts. When a user's request for a service reaches a host, the certificate subject of the user (contained in the proxy) is checked against what is in the local grid-mapfile to find out to which local account (if any) the user certificate is mapped, and this account is then used to perform the requested operation.

2. The second way relies on the ***Virtual Organization Membership Service (VOMS)*** and the ***LCAS/LCMAPS*** mechanism, which allow for a more detailed definition of user privileges, and will be explained in more detail later.

The VOMS is a system that allows for complementing a proxy certificate with extensions containing information about the VO, the VO groups the user belongs to, and the role the user has. In VOMS terminology, a ***group*** is a subset of the VO containing members who share some responsibilities or privileges in the project. Groups are organized hierarchically like a directory tree, starting from a VO-wide root group. A user can be a member of any number of groups, and a VOMS proxy contains the list of all groups the user belongs to, but when the VOMS proxy is created, the user can choose one of these groups as the "primary" group. A ***role*** is an attribute that typically allows a user to acquire special privileges to perform specific tasks. In principle, groups are associated with privileges that the user always has, while roles are associated to privileges that a user needs to have only from time to time. X.509 Attribute Certificates (ACs) [79] are used to bind a set of attributes, like group membership, role, security clearance, etc., with an AC holder. The ***FQAN*** (short form for Fully Qualified Attribute Name) is what VOMS ACs use in place of the Group/Role attributes. They are described in [80]. Grid services are enabled to read VOMS extensions in a proxy and therefore authorize users according to their privileges over a resource.

The ***Local Centre Authorization Service (LCAS)*** handles authorization requests for a service (such as the Computing Element), and the ***Local Credential Mapping Service (LCMAPS)*** provides all local credentials needed for jobs allowed to access and use the resource. Such services are used by the Workload Management system explained later and by other services. The authorization decision of the LCAS is based upon a user's certificate, proxy or extended proxy. The certificate is passed to (plug-in) authorization modules, which grant or deny the access to the resource. A policy file drives such a plug-in.

Another interesting service still in development is the ***Grid Policy Box*** (G-PBox). It allows for the definition of local policies for resources access and usage based on user Grid certificates and VOMS privileges. Through a ***Policy Administration Tool (PAT)*** queries are submitted to a ***Policy Enforcement Point (PEP)***, the entity protecting a resource. The PEP sends requests to a ***Policy***

Decision Point (PDP), which examines the request, contacts the ***Policy Repository (PR)*** to retrieves policies applicable to this request, and determines whether access should be granted according to the rules. The answer is returned to the PEP, which can allows or denies access to the requester.

Information Service

In gLite two implementations of the Information System are used: the Globus MDS, used for resource discovery and to publish the resource status, and the Relational Grid Monitoring Architecture (RGMA) [62], used for accounting, monitoring and publication of user-level information.

MDS with modifications:

Although gLite is based on the MDS, some modifications have been applied to the existing MDS. In particular, the GIIS uses a Berkeley Database Information Index (BDII) to store data, which is more stable than the original Globus GIIS. Finally, a BDII is also used as the top level of the hierarchy: this BDII queries the GIISes at every site and acts as a cache by storing information about the Grid status in its database. The BDII therefore contains all the available information about the Grid. Nevertheless, it is always possible to get information about specific resources by directly contacting the GIISes or even the GRISes.

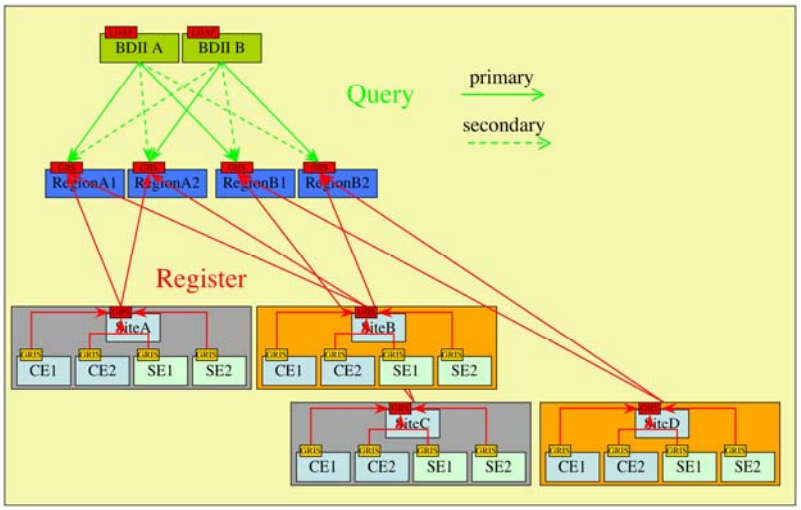


Figure. 2.15 The MDS Information Service in WLCG. The query and register operations are shown. The two top-level BDIIs act as primary and secondary BDII. When the primary is busy, the secondary is used to balance the load.

The top-level BDII obtains information about the sites in the Grid from the Grid Operations Centre (GOC) database, where site managers can insert the contact address of their GIIS as well as other useful information about the site.

R-GMA:

R-GMA is an implementation of the Grid Monitoring Architecture (GMA) proposed by the OGF. In R-GMA, information is in many ways presented as though it were in a global distributed relational database, although there are some differences (for example, a table may have multiple rows with the same primary key). R-GMA presents the information as a single virtual database containing a set of virtual tables. This model is more powerful than the LDAP-based one, since relational databases support more advanced query operations. It is also much easier to modify the schema in R-GMA, making it more suitable for user information. The architecture consists of three major components (cf. Figure 2.16):

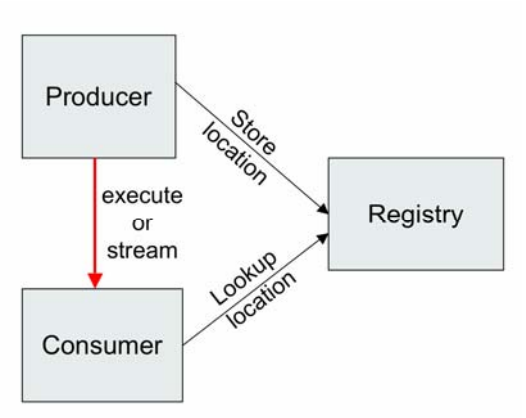


Figure 2.16 The R-GMA Architecture

- The ***Producers***, which provide information, register themselves with the Registry and describe the type and structure of the information they provide.
- The ***Consumers***, which request information, can query the Registry to find out what type of information is available and locate Producers that provide such information. Once this information is known, the Consumer can contact the Producer directly to obtain the relevant data.
- The ***Registry***, which mediates the communication between the Producers and the Consumers.

Note that the producers and consumers are processes (servlets) running in a server machine at each site (sometimes known as a MON box). Users interact with these servlets using command line tools or APIs.

A ***schema*** contains the name and structure (column names, types and settings) of each virtual table in the system. The registry contains a list of producers, which publish information for each table. A consumer runs an SQL query for a table and the registry selects the best producers to answer the query through a process called ***mediation***. The consumer then contacts each producer directly, combines the information and returns a set of ***tuples*** (database rows). The details of this process are hidden from the user, who just receives the tuples in response to a query.

Computing Element
A gLite Computing Element includes a Grid Gate (GG) (equivalent to the Globus gatekeeper), which acts as a generic interface to the cluster via a Local Resource Management System (LRMS) (sometimes called batch system). In gLite the supported LRMS types are OpenPBS, LSF, Maui/Torque, BQS and Condor, with work underway to support Sun Grid Engine.

The gLite worker nodes generally have the same commands and libraries installed as the gLite User Interface, apart from the job management commands. VO-specific application software may be preinstalled at the sites in a dedicated area, typically on a shared file system accessible from all WNs.

It is worth stressing that (in terms of the GLUE schema), a CE corresponds to a single queue in the LRMS. According to this definition, different queues defined in the same cluster are considered different CEs.

Storage Element
gLite supports several different implementations of Storage Elements such as CASTOR [65], dCache [64] or DPM [66]. Furthermore, these Storage Elements can support different data access protocols and

interfaces (see chapter x). Simply speaking, GSIFTP (a GSI-secure FTP) or GridFTP is the protocol for whole-file transfers, while local and remote file access is performed using RFIO [65] or gsidcap [64]. Some storage resources expose a Storage Resource Manager (SRM) interface to provide Grid capabilities like transparent file migration from disk to tape, file pinning, space reservation, etc. However, different SEs may support different versions of the SRM protocol and the capabilities can vary. Disk-only SEs are normally implemented as GridFTP-enabled storage resources which do not have an SRM interface and will be phased out soon in the WLCG infrastructure.

The Disk Pool Manager (DPM) is an SRM-enabled Storage Element. It manages a set of disk servers that users see as a unit of storage space. Mass Storage Systems (with front-end disks and back-end tape storage), like CASTOR, and large disk arrays (e.g. managed with dCache) always provide an SRM interface. Later on in this work we will make a detailed description of storage management in the Grid and specifically in WLCG.

Data Management
The gLite data management client tools allow a user to move data in and out of the Grid, replicate files between Storage Elements, interact with the File Catalog and more. High level data management tools shield the user from the complexities of Storage Element and catalog implementations as well as transport and access protocols. Low level tools and services are also available to achieve specific tasks. One of these tools is the ***File Transfer Service (FTS)*** that allows for the scheduling of the transfer of data files between sites. The service is configured to allow for transfers only on predefined channels between two peers configured at service startup. The FTS uses low level services and tools, such as those available on storage elements to perform data transfer.

Another data management library worth mentioning is the ***Grid File Access Library (GFAL)***. It interacts with Grid File Catalogs and Storage Elements via the SRM interface in order to allow applications to access files using their LFN or GUID. Once presented with an LFN or GUID, the GFAL library contacts the Grid File Catalog (LFC) to find out the SURL of the best replica available. Then, it negotiates with the correspondent Storage Element on the file access protocol to use in order to access the file (POSIX, gsiftp, rfio, gsidcap, etc.).

Workload Management
The purpose of the Workload Management System (WMS) is to accept user jobs, to assign them to the most appropriate Computing Element, to record their status and retrieve their output.

The ***Resource Broker (RB)*** is the machine where the WMS services run. Jobs to be submitted are described using the ***Job Description Language (JDL)***, which specifies, for example, which executable to run and its parameters, files to be moved to and from the worker node, input Grid files needed, and any requirements on the CE and the worker node. The matchmaking process selects among all available CEs those fulfilling the requirements expressed by the user and which are close to specified input Grid files. It then chooses the CE with the highest rank, a quantity derived from the CE status information, which expresses the "goodness" of a CE (typically a function of the numbers of running and queued jobs).

The RB locates the Grid input files specified in the job description using the ***Data Location Interface (DLI)*** [63], which provides a generic standard interface to a Grid file catalog. In this way, the Resource Broker can talk to file catalogs other than the gLite specific file catalog (provided that they have a DLI interface).

The most recent implementation of the WMS allows not only the submission of single jobs, but also collections of jobs (possibly with dependencies between them).

Finally, the ***Logging and Bookkeeping service (LB)*** tracks jobs managed by the WMS. It collects events from many WMS components and records the status and history of the job. Figure 2.17 illustrates the process that takes place when a job is submitted to the Grid.

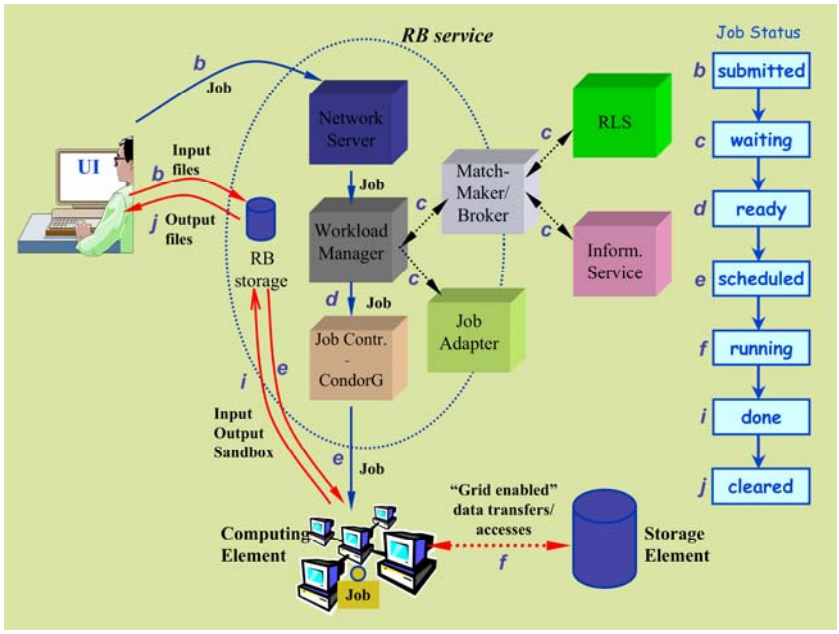


Figure. 2.17 Job flow in gLite. A job is submitted (b); processed by the RB that interrogates the IS and the File Catalogue to find the optimal resources (c-d); sent to the selected CE for execution (e-f); the output is returned to the user (i-j)

2.4.5 ARC

ARC (Advanced Resource Connector) is based on the Globus Toolkit. The idea of developing such a Grid architecture and middleware came after experiencing some shortcomings of GT2. Therefore, the ARC architecture is very similar to the Globus architecture with various extensions concerning job management, brokering, the information system, the GridFTP server and the Globus Resource Specification Language.

Security
As in Globus, the security services are based on GSI. An interesting aspect is how firewalls can be set up with ARC: since both resource management services use port 2811 (the standard GridFTP port), a rather small number of ports needs to be opened with respect to Globus.

Information System
The ARC information services provide static, semi-static and dynamic information about the ARC Grid resources. The information providers are specific scripts that collect dynamic information about the status of the resources and cache it in an OpenLDAP-base database. This functionality is present in the Globus GRIS service that is also used to run of the information providers if the information in cache is stale. A special, simplified usage of the GT2 GIIS OpenLDAP backend allows for building a hierarchical description of Grid-connected sites using an ARC specific schema.

The ARC Grid Monitor client uses a Web browser as an agent to periodically query the distributed information system and to present the results in inter-linked Web pages.

Resource Management
In terms of job management and scheduling, ARC does not use any of the Globus services such as the GRAM. The ARC Grid Manager (partly also responsible for data transfers) running on a computing resource takes care of scheduling jobs, managing the job session directories before and after execution. Through a specialized version of the GridFTP server, data files and executables needed for the job are transferred by the ARC Grid Manager in the job specific session directory and made available to the job. After execution, output and error files are transferred back to the user. ARC client tools and the Grid manager are also able to interact with data services such as the Globus Replica Location Service.

The ARC User Interface is a set of lightweight command line tools to submit, monitor and manage jobs on the Grid, move data around and query resource information. A peculiar feature of the ARC UI is a built-in personal resource broker to select the best matching resource for a job. Grid job requirements are expressed via an extended version of the Globus RSL, xRSL.

Data Access and Management
For what concerns storage management, the ARC development team has developed the so called Smart Storage Element (SSE), a service that provides flexible access control, data integrity between resources and support for autonomous and reliable data replication. SSE uses HTTPs and HTTPg (GSI over HTTP) for data transfer. It interacts with the Globus RLS for file registration and uses GSI for ACL implementation and support. SSE also implements a subset of the Storage Resource Manager (SRM) v1.1 interface as defined by the Grid Storage Management Working Group (GSM-WG) at OGF (see later).

2.5 Current Grid Infrastructures

Previously, we introduced several Grid middleware solutions that are used by different collaborations to build the actual Grid infrastructure. In the following subsection we discuss a selected list of Grid production infrastructures.

2.5.1 Worldwide Large Hadrons Collider Computing Grid (WLCG)

The WLCG [17] infrastructure is the largest in the world, including about 230 sites worldwide [50]. It has been mainly established to support the 4 Large Hadron Collider (LHC) experiments at CERN. The LHC is used to study the fundamental properties of sub-atomic particles and is due to start operating in 2008.

The goal of the WLCG project is to use a world-wide Grid infrastructure of computing centers to provide sufficient computational, storage and network resources to fully exploit the scientific potential of the four major experiments operating on LHC data: Alice, ATLAS, CMS and LHCb. In fact, those experiments will generate enormous amounts of data (10-15Petabytes per year). The processing of this data will require huge computational and storage resources, and the associated human resources for operation and support. It was not considered feasible to concentrate all the resources at one site, and therefore it was agreed that the WLCG computing service would be implemented as a geographically distributed Computational Data Grid. This means that the service will use computing and storage resources installed at a large number of computing sites in many different countries, interconnected by fast networks. The infrastructure relies on several national and regional science Grids, including OSG and NDGF.

The WLCG project was launched in 2003 and is growing rapidly. The Grid operated by the WLCG project is already being tested by the four major experiments that use the LHC to simulate the computing conditions expected once the LHC is fully operational. As a result, the WLCG partners are achieving excellent results for high-speed data transfer, distributed processing and storage. Already, other scientific applications from disciplines such as biomedicine and geophysics are being tested on this unique computing infrastructure, with the support of European funded projects.

WLCG deploys the gLite middleware that hides much of the complexity of the WLCG environment from the user, giving the impression that all of these resources are available in a coherent virtual computer centre. It is built using software coming from various Grid European and American projects, such as Globus (GT2) [14], European DataGrid [16], DataTag [52], GriPhyN [53], iVDGL [54], EGEE [22]. The WLCG infrastructure interoperates with other Grid infrastructures such as OSG and NDGF (see below).

EDG, EGEE and WLCG are among the first projects that have actively and practically tackled the problem of finding a common solution to storage management and access in the Grid, giving life to the Grid Storage Management Working Group (GSM-WG) at OGF.

2.5.2 Open Science Grid (OSG)

The Open Science Grid is a US Grid computing infrastructure that supports scientific computing via an open collaboration of science researchers, software developers and computing, storage and network providers.

The OSG consortium builds and operates the OSG, bringing resources and researchers from universities and national laboratories together and cooperating with other national and international infrastructures to give scientists from many fields access to shared resources worldwide.

The middleware used in the OSG infrastructure is based on the ***Virtual Data Toolkit (VDT)*** [55], a repackaging of the Globus Toolkit, and other services and utilities developed by U.S. Grid projects. The OSG middleware does not provide a full Grid solution but rather a set of basic services needed to build a Grid infrastructure. VDT includes as well a set of virtual data tools to work with virtual data for expressing, executing and tracking the results of workflows. Workflows consist of graphs of application or service invocations, and can be expressed in a location-independent, high-level ***Virtual Data Language (VDL)***. VDL frees the workflow from specifying details of the location of files and programs in a distributed environment. VDL workflows can be executed in a variety of environments ranging from the desktop to Grids such as the Open Science Grid and Teragrid. VDL definitions are stored in a ***Virtual Data Catalog (VDC)*** that provides for the tracking of the provenance of all files derived by the workflow. VDL documents are then interpreted by planner components to generate executable forms of the workflow.

OSG can interoperate with other Grid infrastructure such as WLCG and NDGF through the GLUE schema that present similar description of the resources and specific interGrid gates.

2.5.3 Nordic Data Grid Facility (NDGF)

NDGF [48] is a production Grid facility that leverages existing, national computational resources and Grid infrastructures in the Nordic countries (Denmark, Norway, Sweden and Finland).

Currently, several Nordic resources are accessible through the Advanced Resource Connector (ARC) middleware and gLite [49] Grid middleware.

Chapter 3

THE CHALLENGE OF DATA STORAGE IN WLCG

In this chapter we first outline the overall distributed computing model chosen by LHC experiments. We then focus on the data and storage requirements in order to provide the necessary background for discussing operational issues and requirements of distributed worldwide Grid storage solutions.

3.1 Introduction

Although there are different HEP experiments within the WLCG project, all of them follow a certain way of organizing the basic distributed computing model. We first describe the general computing and data model that is applicable to all four experiments and outline experiment specific differences later whenever necessary.

3.1.1 Constraints for Distributed Computing and Storage

All four experiments have the following items in common which can be regarded as the main constraints for a distributed computing model:

- **Central data recording**: Raw data are produced in a central location. In particular, particle detectors located at CERN record particle interactions and tracks that appear during a particle collision. Data is typically only written once but never updated (i.e. **read-only data**).
- **Large data storage**: Each experiment produces a few (5 to 10) Petabytes of data each year that need to be stored permanently.
- **Data processing**: Raw data needs to be processed in order to extract and summarize information that has relevance to physics. This reprocessing of data is called ***reconstruction*** in HEP terminology and is typically very computing intensive. The storage requirement for reconstructed data is smaller than for raw data but still in the order of many Terabytes to a few Petabytes per year [67][68][69][70].
- **Distributed computing and storage centers**: CERN is considered to be the main center to provide storage and processing power. However, each of the 4 experiments forms a collaboration of many countries where almost each of the countries provides storage and computing capacity that is dedicated to one or more physics experiments. In this way, the overall computing power as well as the overall storage capacity available to a single experiment is increased.
- **Distributed user community**: Not only computing and storage hardware is distributed but also the user community. In fact, many hundreds of research institutes and universities participate in CERN experiments with physicists (the actual end users) distributed all over the globe. Their common goal is to analyze physics data from remote locations, still having transparent access and good response time when accessing data in different locations.

3.1.2 Data Model

We now go into the details of an experiments data model, i.e. we analyze how data is organized and stored in different storage devices. We start our discussion with a simplified model of how data is

generated, processed and stored. Each of the LHC experiment describes its data model in a very detailed report, called the Technical Design Report (TDR). All experiments TDRs can be found in [67][68][69][70].

In brief, particles are first accelerated in the Large Hadrons Collider (LHC) before they collide in the four particle detectors of the four different LHC experiments. Each particle collision within the detector is called an *event* in physics terminology. During such a collision new particles are created and go through several layers in the detector. Since collisions appear with a high rate (they are experiment specific), specialized hardware is used to reduce the number of physics events to record (cf. Figure 3.1). This is typically called event "filtering". In addition to these hardware filters, there are also software filters that further select interesting physics events which are then stored in raw format (also referred to as **raw data**) on storage devices to keep the data permanently available. This first filtering of data and recording of raw data is typically referred to as the *"on-line computing system"*. The raw data recording rate varies between 0.2 and 1.25GB/sec. In summary, the on-line computing system filters and stores the physics events, which then have to be processed by the off-line system (see below). Since raw data only contains the most basic information of the collision, a re-processing step is required which then allows for the actual analysis of the data. This reprocessing is done by the so-called "*off-line computing system*" after the raw data has been stored to a safe location. This reprocessing (also referred to as reconstruction step) is very computing intensive since several physics interactions need to be considered.

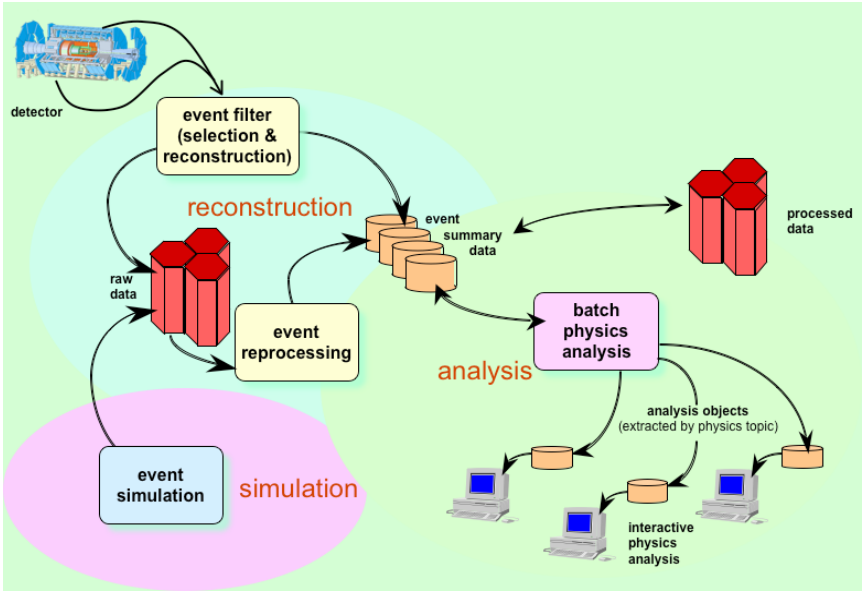


Figure 3.1 Data flow in a typical HEP experiment

Finally, new data is created which is typically called **reconstructed data**. This type of data is of smaller size than the raw data and is typically stored on random access storage devices such as disks whereas raw data is often stored on tape systems.

The on-line and the off-line computing systems have very different software requirements. Whereas the on-line system requires high-performance real-time systems that work on local area networks, the off-line system takes care of the actual storage and distribution of the data. In the remainder of this thesis we will only discuss the storage challenge of the off-line system and how Grid approaches can be used to tackle the problem.

In order to allow for an effective and efficient physics analysis of reconstructed data, summary information of reconstructed data is stored in another type of data referred to as **event summary data** (ESD) or simply **tag data**. This type of data contains summary information of most important physics events stored in reconstructed data.

In order to predict the behavior of collisions and therefore confirm a physics model, simulated physics events need to be produced and stored and then processed, as it happens with real data. Simulated data are produced from a detailed Monte Carlo model of the physics detector that incorporates the current best understanding of the detector response, trigger response and "dead" material. Besides simulated events, this data contain also extra information regarding the history of the event. Simulated raw data sets are thus larger than real raw data. The simulation task is also very computer intensive and takes place generally at smaller centers, in order to allow big centers concentrate on real data processing. The resulting amount of data is comparable to that of the real data and totally is as well of the order of few Petabytes. Samples of this data could be accessed later on for physics analysis.

In summary, high energy physics experiments are characterized by three types of data (all read-only) with different characteristics and data sizes. Since each of the four experiments use different software tools in the reconstruction process and/or analyze different physics behaviors, the data sizes are not identical but we give a rough estimate below:

- Raw data: data size about 1-2 MBs per physics event.
- Reconstructed data: data size: 200 – 500 KBs per event
- Event summary data: a few tens to hundred bytes per event
- Simulated data: about 1-2MBs per physics event.

3.1.3 Storage Systems and Access Patterns

Now that we have a brief overview of the main storage requirements, let us have a closer look at the storage systems that are used to store the different data types. Due to the large storage requirement for both raw and reconstructed data (every year about 10 Petabytes of additional storage space are required), it is not economically feasible to store all data on disks. Therefore, hierarchical storage systems with both disk and tape storage media are used to archive the majority of data. Typical examples are CASTOR (add ref), HPSS, diskExtender etc. that manage both disk pools and tape libraries in order to provide for different access patterns:

- Fast, random read/write access to data stored on the disk cache.
- Slower, but sustained sequential read access to data permanently archived on tape systems.

Physics data described in Section 3.1.2 is characterized by different *access patterns*, which can also be mapped to certain "storage types". In general, after the reconstruction step (that can be executed multiple times and with different algorithms), raw data is accessed very infrequently and therefore can be mainly stored on tape systems. In case read access is required, the data can be staged to the disk pool of the hierarchical storage system and users can access it. In contrast, reconstructed data and in particular event summary data need to be accessed very frequently and therefore need to be "always" available for random access. Consequently, these two types of data are typically stored on disk rather than tape.

In practice, hierarchical storage systems are rather expensive, and small computing centers often cannot afford such systems. In these cases, disk pools (connected via a shared file system or other storage technologies that can provide a single name space) are used.

In summary, based on the size and the role of the computing center within the collaboration (details will be given in the next section), different storage solutions with rather different characteristics are used. In general, each computing center can deploy the software and hardware system of its choice to serve data to the end user. In principle, this model is fine if people only access data locally within a site. For instance, user A located at site A1 accesses her files served by Castor whereas user B located at site B1 access her files via GPFS (a parallel file system). However, as soon as data needs to be transferred between sites (i.e. independent administrative domains) interoperability issues arise since both storage systems need to support the same file transfer protocols.

Given the huge amount of data HEP experiments deal with, sometime it might be practical to transfer very significant samples of data by shipping physical media (tapes or DVDs). Then, depending on the software used to write the media, specific import/export procedures and protocols need to be agreed in order to be able to read back the data. This is for instance the strategy that CERN has adopted for some time with CASTOR or its predecessor SHIFT. Export utilities are available to write data on tape using the standard protocol ANSI v3 for tape labeling. This has allowed many experiments in the past to import CERN tapes into other systems (such as YASS [71] for the NA48 experiment) and serve automatically and transparently the data on them to physicists on demand. However, this practice has significantly slowed down data availability and access for researchers, which is crucial when pursuing a physics result.

Other issues concern the *heterogeneity* of sites, both in terms of hardware and software: sites can use different platforms, operating systems, compilers, etc. This also comprises conventional portability issues such as hardware with little or big endian architectures. The latter issue is not only important in terms of networking but particularly when storing integer or floating point variables.

3.2 Multi-Tier Architecture

Based on the computing constraints and the data model presented in Section 3.1, CERN experiments have established a distributed computing model that is based on a hierarchical, multi-tier architecture as depicted in Figure 3.2. In particular, CERN (with its computing center and the particle detectors) is regarded to be the Tier 0 center in the architecture.

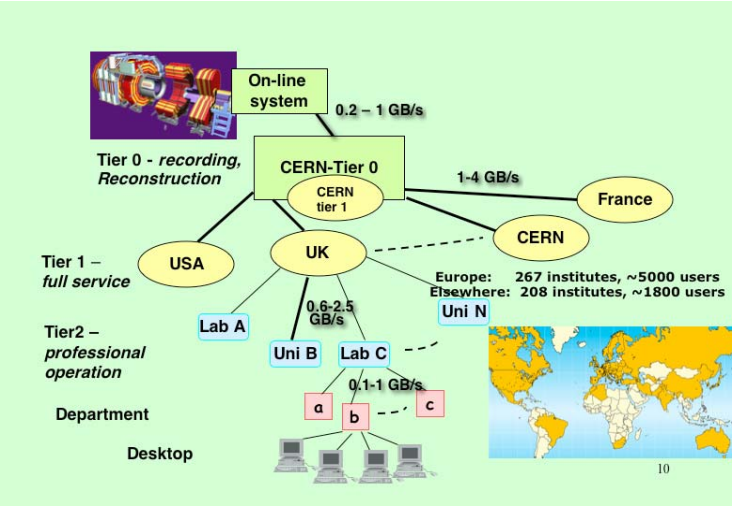


Figure 3.2 Multi-tier architecture of distributed computing centers

The *Tier 0* center is not only the root in the hierarchical system, it is also the main data store for all raw data created by the for LHC experiments. Additionally, a large number of reconstructed and simulated data is stored there. In general, CERN provides about 30% of the total needed storage capacity both for disk and tape storage. Figure 3.3 shows a summary of resources provided by CERN. Figure 3.4 shows as well the resources required by each of the experiments in 2008 (start of LHC operations) and how much it will be provided by CERN.

CERN Tier0	2004	2005	2006	2007	2008	2009	2010
CPU (kSI2K)			2400	4800	12500	15900	26200
Disk (Tbytes)			230	450	1260	1330	1770
Tape (Tbytes)			1500	3400	13600	23600	33900
Nominal WAN (Mbits/sec)	30000	60000	80000	100000	120000	140000	160000

CERN Analysis Facility	2004	2005	2006	2007	2008	2009	2010
CPU (kSI2K)			2600	5100	10000	14600	15000
Disk (Tbytes)			850	1700	4200	5200	5300
Tape (Tbytes)				1500	3400	5700	5900

Table 3.1 CPU, Storage and Network resources provided by CERN (Tier 0).

At the next tier, the so-called "*Regional Centers*" (also referred to as *Tier 1* sites) are located that provide main computing and storage facilities for certain regions in Europe, North America and Asia Pacific. One experiment typically can use between 5 to 10 Tier 1 sites of the total 11 available. For example, the CMS experiment has Tier1 sites in Barcelona (Spain), Bologna (Italy), Karlsruhe (Germany), Oxford (UK), Lyon (France), Batavia (Illinois, USA) and Taipei (Taiwan).

CERN Tier0	Split 2008	ALICE	ATLAS	CMS	LHCb	SUM 2008
CPU (kSI2K)	Required	3300	4060	4600	570	12530
	Offered	3300	4030	4600	570	12500
	% of Req.	100%	99%	100%	100%	100%
Disk (Tbytes)	Required	240	350	400	270	1260
	Offered	240	350	400	270	1260
	% of Req.	100%	100%	100%	100%	100%
Tape (Tbytes)	Required	2480	5680	4900	500	13560
	Offered	2500	5700	4900	500	13600
	% of Req.	101%	100%	100%	100%	100%
Nominal WAN (Mbits/sec)						

CERN Analysis Facility	Split 2008	ALICE	ATLAS	CMS	LHCb	SUM 2008
CPU (kSI2K)	Required	5000	2650	4800	330	12780
	Offered					10000
	% of Req.					78%
Disk (Tbytes)	Required	1450	1850	1500	450	5250
	Offered					4200
	% of Req.					80%
Tape (Tbytes)	Required	1160	520	1900	860	4440
	Offered					3400
	% of Req.					77%

Table 3.2 CPU, Storage and Network resources provided by CERN (Tier 0) per experiment. CPU power is expressed in Kilo SpecInt2000, while disk and tape capacity is expressed in Terabytes.

Tier 1 centers have very good wide-area **network connectivity** to the Tier 0 of the order of 1-2 GB/s. At the moment of writing, during the WLCG service challenge, an aggregate bandwidth of about 2 GB/s has

been measured from CERN to the Tier-1 centers. In 2010, CERN should provide an aggregate bandwidth of about 16GB/s.

Furthermore, each of the Tier 1 centers stores a fraction of raw data as well as reconstructed data. The latter is typically replicated to several sites, i.e. redundant data copies are available for both improved read access latency for local users in a region as well as for fault tolerance. Figure 3.5 shows a summary of resources provided by the 11 Tier-1s centers from 2004 to 2010. It shows as well the situation of the resources available to the 4 LHC experiments in 2008, when LHC operations will start.

Summary Tier1s	2004	2005	2006	2007	2008	2009	2010
CPU (kSI2K)	1205	4940	9504	19372	46732	65722	93425
Disk (Tbytes)	264	1225	3715	9532	23349	32233	45842
Tape (Tbytes)	323	1665	4153	8038	22368	35563	50745

TDR Requirements 2008	ALICE	ATLAS	CMS	LHCb	SUM
CPU (kSI2K)	12300	24000	15200	4400	55900
Disk (Tbytes)	7400	14400	7000	2400	31200
Tape (Tbytes)	6900	9000	16700	2100	34700
Number of T1s	6	10	7	6	n/a

Table 3.3 CPU, Storage and Network resources provided by the 11 Tier-1 sites.

In each of the regions served by Regional Centers there are typically several research institutes and universities (Tier 2 sites) that are connected to Regional Centers. This does not necessarily mean that there is a dedicated network connection between the sites, but they should have sufficient network connectivity to be able to transfer data between Tier 1 and Tier 2 and vice versa. Generally, the wide-area network bandwidth should be in the order of a few 100 MB/s. As a result, Tier 2 sites serve both data and computing resources to physicists working in their home university without the need to travel to CERN for data analysis. Tier 2 sites are furthermore involved in the production of simulated data. Such data is optionally and only partially stored at Tier 2 centers, the entire simulation production being available and sometime replicated between the Tier 1 centers. Figure 3.6 shows the expected resources provided by all Tier 2 centers by 2008.

Tier-2 Planning for 2008		ALICE	ATLAS	CMS	LHCb	SUM 2008
CPU - MSI2K	Offered	5.6	20.0	17.4	4.4	47.4
	TDR Requirements	14.4	19.9	19.3	7.7	61.3
	Balance	-61%	0%	-10%	-42%	-23%
Disk - PBytes	Offered	1.5	6.2	4.5	0.8	13
	TDR Requirements	3.5	8.7	4.9	0.023	17.1
	Balance	-58%	-29%	-8%	n/a	-24%
# Tier-2 federations - included(expected)		13 (14)	22 (29)	17 (20)	10 (11)	30 (39)

Table 3.4 CPU and Storage resources provided by the Tier-2 centers in 2008. The experiment requirements as expressed in the respective Technical Design Reports are also reported and the difference with respect to the expectations marked as red.

Finally, institutes or universities might have very small computing/storage facilities that are the Tier 3 resources. These facilities do not necessarily provide an essential infrastructure, but might be important in case local resources at Tier 2 sites become scarce.

3.2.1 Discussion of the Tier Model

The hierarchical tier model with different minimum networking requirements and replicated data allows for both fast data access of users located in remote sites and a high degree of fault-tolerance in terms of data and computing availability. The overall vision is that access to data and computing resources should be transparent to the user.

Although the idea of using a tier-based distributed computing model was already introduced in the early 1990s when the LHC experiments where designed, the actual realization of the overall vision of a transparent computing infrastructure could only be made possible with the advances in networking technology. In the 1990s it was often impossible to transfer the large amounts of physics data between Regional Centers. For example, with a wide-area network link with a hardware bandwidth limit of about 100 Kbps one could only transfer about 1 GB of data per day: that was not enough to satisfy the requirements of Regional Centers. Therefore, data was not transferred over the network but first exported to tapes which were then shipped via plain mail to Regional Centers which then imported the data locally.

On the other hand, modern wide-area network links that are used today (i.e. in 2006) have capacities that range from 100 MB/s to several GB/s. Therefore, the amount of data to be transferred is more than 1000 times larger than in the 1990s. Consequently, the data transfer problem can now be solved.

As a result, **common interfaces** to local storage systems have become more and more important since data transfer has become feasible. In particular, storage systems must provide for common protocols to all users located in different Regional Centers and the standardization work of the Storage Resource Manager (SRM) interface is an important contribution in this direction.

3.3 High Level Physics Use Cases

In the following section we give an overview of the basic high-level use cases for the computing usage in High Energy Physics. These use cases are very much representative of the data model explained in the previous section as well as of a specific experiment. For simplicity, we take the CMS experiment as an example [69] but assume that this brief discussion is representative for other experiments, too.

3.3.1 Reconstruction

The raw data, whether real or simulated, must be reconstructed in order to provide physical quantities such as calorimeter clusters, position and momentum of particles, information about particle identification and so on. The pattern recognition algorithms in the reconstruction program make use of calibration and alignment constants to correct for any temporal changes in the response of the detector and its electronics, and in its movement. This process is computationally very intensive and needs to be repeated a few times in order to accommodate improvements in the algorithms, in calibration and alignment constants. It requires about 100 Million SI2000 [72] (about 50,000 of today's PCs) per year for the 4 experiments. Therefore, it cannot be executed entirely at CERN. Raw data are stored on tape at

CERN and streamed to Tier 1 sites where an automatic machinery is in place to start the reconstruction program on the data just arrived. For this use case the storage requirements are the following:

- Specific data transfer servers with *WAN access* and adequate large buffers need to be in place in order to efficiently receive data coming from the Tier 0.
- *Discovery functions* should allow for the identification of the data services and buffers dedicated to the given experiments.
- Data transfer services should allow for *reliable* and *secure transfer* of big buffers of data. Such services should provide users with transfer scheduling and retry functionalities.
- Data transfer servers must be connected to the *tape storage* systems for persistent storageof the data.
- A proper *storage interface* to mass storage systems should be available in order to trigger and control store and stage operations in an implementation independent way.
- Given the amount of data involved, it is desirable to avoid making multiple copies of the data. Therefore, the data need to remain on disk for a time sufficient to reconstruct them, before they are deleted to make space for new data. The *"pinning" functionality* allows for specifying a lifetime associated to the data stored in a given space.
- For a critical operation such as reconstruction of physics data, it is mandatory not to compete for resources with other experiments. Therefore, *dedicated resources* are normally required by the experiments.

3.3.2 Main Stream Analysis

This use case can be considered as the standard, scheduled activity of a physics group in a certain university. The research group is interested to analyze a certain data set (typically consisting of several Giga- or Terabytes of data) in a certain Tier 1 center that has free computing capacity. If the data are not available in that site, they need to be transferred in a scheduled way and the operation might last for a few days. Once the data has arrived, computing-intensive physics analysis operations can be done on the specified data. For instance, the validation of reconstructed data is a process in which the validity of certain algorithms is verified. This process implies access to 1-2% of the total reconstructed data of an experiment. It might imply rerunning variations of the program several time on the same set of data. Once the process is finished, the result is stored on tape.

The implicit storage, data and transfer requirements are as follows:

- Data need to be accessible at the storage system, i.e. mass storage systems, disk systems as well as the corresponding data servers need to provide the required performance.
- Data transfers tools need to be in place that have access to the source storage system and can transfer data to another storage system at a different site/tier. Since the physics activity and therefore also the data transfer are scheduled, the data transfer can be optimized in order to allow for a maximum usage of certain network links: bandwidth can be "reserved" by *prioritizing* this particular physics group and reducing the data transfer of other physics groups or individual users.
- Once data has arrived at the site, computing and storage *resources* must be dynamically or statically *reserved* for a particular user group.
- It should be possible to express *ownership of resources* and specify *authorization patterns*.
- In order to ensure resource sharing, *quotas* should possibly be enforced in a transparent way so that several groups within the VO or even multiple VOs can concurrently use the resources at a site.

- Resources usage and *status* should be *monitored* and published so that busy resources are not selected for further computing and/or storage tasks.
- If the needed data are on tape, they must first be transferred to disk for online access. Therefore, transparent staging tools must be available.
- Specific *file access protocols* need to be supported by the storage facility so that applications using only those protocols can be executed.
- Once data are analyzed, the relative output can be saved on tape if considered to be correct and of value for further processing. Therefore, tools to archive the result on tape and register the result in the Grid are necessary.
- Physicists should be provided with the necessary tools to *manage space*, for instance in case the storage system does not remove unneeded files automatically.

Grid operators and/or site administrators that take care of the execution and monitoring of data transfers as well as the allocation of CPU power to the physics group can further support and optimize the actual execution of this use case scenario.

3.3.3 Calibration Study

During the run of the LHC, particles pass through detectors that have to be aligned and calibrated in order to allow for correct physics analysis. A physics group might work on the calibration study and detect problems with the calibration and alignment constants. In such a case, some of the reconstruction algorithms need to be rerun and new reconstructed data needs to be stored.

In this use case, there is a high request for fast access to a subset of data as well as large amount of computing power at certain peak times. This might also involve transferring raw data to disk caches or disk pools to allow reconstruction algorithms to reprocess existing data. This could be a very costly operation since it can schedule transfers of huge amounts of data from tape to disk. Many tape drives can be busy in this task that might have high priority. Once the calibration constants obtained prove to be accurate, they are stored in experiment specific databases that are distributed to a few sites for performance and fault tolerance reasons.

3.3.4 Hot Channel

This type of analysis is also sometimes called "chaotic" analysis. In contrast to the scheduled "main stream analysis" (Section 3.3.2) of a particular physics group, here a single physicist working on some very specific analysis might request data access which can be of "any" size, i.e. it is not known a priori how much data would need to be made available in disk pools or transferred over the wide area network.

This use case is of particular importance for physicists, system administrators, operators, and developers since it can create particular worst-case scenarios that stress the system. This use can also help detect scalability issues in many parts of the data access and storage system. For instance, how many files can be requested to stage from tape to disk without seriously degrading the performance of the system? How many files can be transferred in parallel between two or more sites without seriously effecting scheduled activities mentioned in Section 3.3.2? How robust are file catalogues?

Because of this chaotic and unpredictable behavior, it is very important to be able to control storage resource usage and access accurately in order to prevent problems. In particular, quota and dynamic space reservation become essential. Additionally, the ability to control data and resource access establishing

local policies and access control lists is fundamental. For instance, the capability of staging files from tape to disk or to store results permanently on tape should be allowed only to users covering certain roles and belonging to specific groups. VO managers are allowed to check periodically the resources available to them to ensure correct usage. They need to check for file ownership, correct placement of files, file sizes, etc. VO managers can delete files or move them to storage with appropriate quality whenever needed.

3.4 Grid Data Access in Practice – The Client Side

After we have seen the data model, the distributed storage resources and the basic high level physics use cases, let us now consider how a physicist can actually access data in the Grid as required in the high level physics use cases. This will help us elaborate on data access protocols and technologies in later chapters.

Our guiding example is a physicist accessing data with a physics analysis program in order to create a histogram of some particle interaction. In order to do so, the physicist needs to have:

1. A client library (physics code) that reads data, does some calculation and then produces a histogram. For simplicity, we neglect the time it takes to plot the histogram but only consider the basic file operations on read-only data, i.e. we reduce the discussion to the following basic file operations: **open**, **read**, **lseek**, **close**. These file operations are used by the physics client library by making (native) systems calls to the file system.

2. A physics dataset (typically reconstructed data) that is organized in a way that the physics client library mentioned above can extract useful information. The dataset itself can be contained either in a single or in multiple files.

Our physicists has now several possibilities to access the data and plot her histogram:

1. **Local access with local file system**. In the simplest case she has a desktop or laptop computer where the physics client library is installed and the physics dataset is available on a local disk. In this case, the physics library uses the standard UNIX file system calls (either via static or dynamic libraries) to open a local file and read the required bytes. She has full control over the entire system and is mainly limited by her local disk space and disk performance.

2. **Local access with a distributed file system**. In case the data sets become too big (we have seen that reconstructed data can have several Tera- or even Petabytes), a single hard disk cannot store all the required data. Therefore, a distributed file system is used that can manage several disk volumes but still offers a single name space for files and provides for transparent access to files located on different disks. In this way, our physicist can use the physics library in the same way as in scenario 1 with a local disk. Typically, distributed file systems are operated over low latency local area networks.

   Distributed file systems provide a very good abstraction for the end user but sometimes have too long read and write latencies in case of several users (in our case a big physics group) accessing many files. The main file server might get overloaded. In order to provide for better read/write performance, *parallel file systems* are used that provide several data copies (replicas) and several file servers which can access parts of a file in parallel. More details on distributed and parallel file systems are given in Chapter 4.

3. **Local access with a mass storage system**. Let us assume that our physicist is not only interested in analyzing reconstructed data but also needs access to some parts of the raw data. As we have seen earlier in this chapter, raw data is mainly stored on tape and managed by mass storage systems since not even distributed nor parallel file systems can provide a cost-effective solution to the data management problem of LHC experiments. However, this situation creates a first "problem" or better a "challenge" since access latencies between disks and tape systems are not necessarily comparable. Additionally, some raw data might have been accessed previously by another physicist and therefore still reside in a disk cache whereas another portion of the raw data required by our physicist is on a tape drive that has not yet been mounted and needs several seconds in order to be accessible.

   At this point, the physics analysis code cannot be just used in the same way as with a conventional file system approach since the mass storage system imposes additional access latencies that end users must deal with explicitly. Therefore, users of mass storage systems need to be aware that client code might have very different performance results with certain datasets. It might also become very difficult to estimate the actual performance (observed wall clock time) of the physics code without some details about the storage locations of certain reconstructed and raw data sets.

4. **Remote access.** In all the previous 3 examples we have assumed that standard POSIX file systems calls are used to access files located in either local or remote storage systems. However, in many cases data stores or object storage systems do not provide a standard POSIX interface and/or provide an additional interface with a specific file access protocol. For example, the most common physics analysis and data access system is ROOT [73] that provides an object-oriented interface to storage. In this case, simple POSIX I/O is not sufficient on the client side, and our physicist needs to use the particular ROOT interface to write and read data. However, ROOT itself uses POSIX file system calls when storing and retrieving data but provides a much higher-level interface to the end user. Additionally, ROOT has its own data access protocol to access data stored on remote disks. Further details on the ROOT remote access system via xrootd will be given in a later chapter.

   The main point about this remote access use case it that high level language interfaces (sometimes proprietary ones) are required and end user applications needs to be adapted to that if not originally written with such an interface. Other examples will be mentioned and discussed in various chapters in this thesis.

5. **Grid access (separation of computing and storage)**. Let us now consider a more complex use case where our physicist needs to run a long calculation on her data set. In this case she would submit the executable to a Data Grid, which provides both computing power and storage. However, the typical data access model in a Grid environment is not necessarily identical to the data access model provided by a file system. In Grid systems (in particular in the current version of the WLCG project) computing resources such as clusters and storage systems are not always connected via direct data access methods. In particular, data available on Grid storage systems is not always directly available on the worker nodes of clusters through mounted filesystems. That means that direct POSIX file system calls cannot be used to access the data. This is in particular true if a large dataset is available in Regional Center 1 whereas the physics program runs on a cluster in Regional Center 2. In such a case data either needs to be transferred to the local worker node or remote access libraries need to be used to open the file and read/write its contents.

   Given this separation of storage and computing and the fact that jobs not always run in locations where all required data are present, there is a high demand for remote access methods as well as

for file transfer tools to transfer data from remote storage systems to local worker nodes. In particular it is important to guarantee standard POSIX file access even in case of remote I/O where calls to the local file system can be trapped and redirected to remote sites. A major use case for this is that proprietary code cannot be changed but should still be made Grid-aware *without* changing the way file access is done within the application.

Another approach to this problem is the use of logical and physical filenames. In detail, the client application would only work with unique logical filenames whereas it is up to the Grid system to provide a physical file instance to the application program. This approach has the advantage that the Grid middleware can locate files via replica catalogs, choose a particular physical instance of a file replica and transfer it to the client (located on a worker node in a cluster). This also requires special purpose client libraries for open, read, write etc. that need to be used instead of conventional POSIX open, read, write operations on a UNIX file system. The latter approach is provided by a library called GFAL [75] in WLCG and will be discussed later in this thesis.

A practical problem to this use case and the general remote data access use code in scenario 4 is that some clusters do not allow for outbound network connectivity on worker nodes, i.e. **firewall settings** restrict the way data access is achieved. In such a case, random data access to Grid storage at runtime is either not possible or needs to be redirected to local data servers that can act as proxies to provide access to remote data.

The five different use cases above outline the main usage of data access in a Grid environment and are therefore important in the remainder of this thesis when we talk about different storage and protocol aspects. One might argue that a distributed operating system should be in place that spans all sites in all tiers of the WLCG computing model. Potentially, this can solve the problem of transparent data access intrinsically. In fact, there are several research groups that work in that direction but actual implementations (in particular within WLCG) have not yet reached this high level goal, and therefore storage management and access is still a challenge.

3.5 Storage Requirements

In the second quarter of 2005 the WLCG Baseline Service working group [76] has been established in order to understand the experiment requirements for the first data challenges. A data challenge is a set of tests focused on verifying the readiness and functionality of the computing frameworks of the LHC experiments. The report of such a group has been published [77]. In such a report the main functionalities of a storage service, and the needed file access protocols are listed together with target dates for their availability.

At the end of 2006, a second working group was established in WLCG, the Storage Class WG. The mandate of this working group has been to understand definition and implementation of storage qualities demanded by the experiments.

In what follows we summarize what has been requested by the experiments for the beginning of the LHC activities. In the last part of this chapter we also give an overview of the results achieved by the Storage Class WG.

3.5.1 Overview

A Storage Element (SE) is a logical entity that provides the following services and interfaces:

- A mass storage system (MSS) that can be provided by either a pool of disk servers or more specialized high-performing disk-based hardware, or disk cache front-end backed by a tape system. A description of mass storage management systems currently in use in WLCG is given in the next chapter.
- A storage interface to provide a common way to access the specific MSS, no matter what the implementation of the MSS is. In particular, the minimum set of functions and services that a storage system must provide in an MSS-implementation independent way is defined later in this section.
- A GridFTP service to provide data transfer in and out of the SE to and from the Grid. This is the essential basic mechanism by which data is imported to and exported from the SE. The implementation of this service must scale to the bandwidth required. Normally, the GridFTP transfer will be invoked indirectly via the File Transfer Service (cf. Chapter 2) or via the storage interface.
- Local POSIX-like input/output calls providing application access to the data on the SE. The available protocols and libraries used by LHC applications are described later in this thesis. Various mechanisms for hiding this complexity also exist, such as the Grid File Access Library in WLCG. These mechanisms include connections to Grid file catalogues to enable to access a file using the Grid LFN or GUID (cf. Chapter 2).
- Authentication, authorization and audit/accounting facilities. The SE should provide and respect ACLs for files and data-sets, with access control based on the use of extended X.509 proxy certificates with a user DN and attributes based on VOMS roles and groups (cf. Chapter 2). It is essential that an SE provides sufficient information to allow tracing of all activities for an agreed historical period, permitting audit on the activities. It should also provide information and statistics on the use of the storage resources, according to schema and policies to be defined.

A site may provide multiple SEs with different qualities of storage. For example, it may be considered convenient to provide an SE for data intended to remain for extended periods and a separate SE for data that is transient – needed only for the lifetime of a job or set of jobs. Large sites with MSS-based SEs may also deploy disk-only SEs for such a purpose or for general use. Since most applications will not communicate with the storage system directly, but will use higher-level applications such as ROOT [73], it is clear that these applications must also be enabled to work with storage interfaces.

3.5.2 The Storage Interface

It is fundamental that a homogeneous interface to mass storage systems is provided and deployed at all sites to allow for smooth Grid operation and uniform behavior. Therefore, the WLCG baseline service working group has defined a set of required functionalities that all SE services must implement before the start of LHC operations. The uniform storage interface agreed must therefore be implemented by all storage services part of WLCG. In what follows we describe the main characteristics of the Storage Interface.

File types
The need to introduce the concept of several file types has been recognized:

- ***Volatile***: it is a temporary and often sharable copy of an MSS resident file. If space is needed, an unused temporary file can be removed by the MSS garbage collector daemon, typically according to the Least Recently Used (LRU) policy. Applications using the file can impose or extend the lifetime of the file ("pinning"). If the lifetime is still valid a file cannot be automatically removed by the system. However, in case space is needed, storage systems may choose to remove unused files, even if pinned.
- ***Durable***: it is a file that cannot be removed automatically by the system. A user may assign this type for a new file if he/she does not know yet if the file should be copied to MSS. If the disk is full and space is needed, the local implementation may decide to copy the file to MSS or to send a mail to a VO admin. Such manual interventions are generally not encouraged.
- ***Permanent***: it is a file that can be removed only by the owner or other authorized users.

The default file type is "Volatile". Users can always explicitly remove files. However, at least at the start of LHC, the experiments want to store files as permanent. Even for really temporary files experiments will be responsible for the removal operation. Higher-level middleware should enforce the permanent file type for LHC VOs.

Space types
It was expressed the need to be able to operate on a set of files at once. This can be achieved introducing the concept of space. A logical space is a container that hosts a set of files. However, there are different types of files. Therefore, the concept of space type must be introduced as well. Three categories of space have been defined corresponding to the three file types. The meaning of each type being similar to the one defined for files. A logical space is connected to a physical space since a site may decide to offer different Quality Of Service for the 3 categories, for example better disk hardware for "Permanent" than for "Volatile". Normally a file of a certain type resides in a space (container) of the same type. But if the file type of one file is changed from "Volatile" to "Permanent", the site has the freedom to move the file to the "Permanent" space container or to keep it "Permanent" in the "Volatile" space container.

Space reservation
Experiments require the ability to dynamically reserve space to ensure that a store operation does not fail. Space can be statically reserved by site administrators explicitly allocating storage hardware resources to a VO, or in advance by VO managers (via some administrative interface), or at run time by generic VO users. VO users should be able to reserve space within spaces already allocated by VO managers or by site administrators for the VO (or for a specific VO group).

The reservation has a lifetime associated with it. The user is given back a space token that he/she will provide in the following requests to retrieve a file from tape or from another remote storage system, or to write a new file. When the space reserved has been exhausted, the next requests will fail with "No user space". Space reservation should support the notion of "streaming" mode: when space is exhausted, new requests will not fail but will simply wait for space released by other users.

Global space release
The space can be released either by the user/VO manager or by the system when the lifetime associated with the space expires. There are two modes for space release: by "default", permanent files are kept, while volatile files are kept until the Pin time expires; if the "force" mode is specified, all files even permanent or pinned are removed. Other needed functions that deal with space management are:

- A function to change the amount of space reserved (the size of the container) and/or the space lifetime.

- A function to reclaim the space occupied by the released files (but the released files are not necessarily removed from disk) and update space size to current usage.

Permission functions
Similar to POSIX Access Control Lists, permissions may be associated with directories or files. In fact, Unix-like permissions are not enough to describe proper authorization patterns. LHC VOs desire storage systems to respect permissions based on VOMS groups and roles. ACLs are inherited from the parent directory, by default. Initially, at the start of the LHC, file and directory ownership by individual users are not needed, even though it must be possible to track down all operations executed by individuals. It is necessary however to distinguish production managers (people with proper privileges to execute reconstruction or calibration programs) from unprivileged users. This is needed in order to be able to control write access to dedicated stager pools or to "precious" directories. This is important and must be supported by all storage interface implementations.

Directory functions
For administrative reasons functions to create/remove directories, delete files, rename directories or files are needed. Such functions have sometime caused problems in the past, as in the case of the "remove" function. In fact, some storage system implements the remove function as an "advisory delete": the file is not physically removed but it is marked as removable. Only when space is needed the file is then removed also from the namespace. This has caused problems when other files with same name but different/correct content were created to replace the files removed. File listing functions are also needed. A directory listing some time can become really long, therefore implementations can choose to truncate the output to an implementation-dependent maximum size. Full recursive listing of directories is also possible but this operation can complicate both client and server implementations. Cookies can be used to return list chunks, but this will make the server stateful. It is advised to avoid large directories. Directories in a managed storage system are namespace implementations. There is no need for "mv" operations between two different SEs.

Data transfer control functions
These functions do not normally move any data but prepare the access to data. These are the functions that deal with the MSS stagers. The only exception is the copy function that moves data between two Storage Elements. All these functions can operate on a set of files and they report a status or error condition for each file. The functions are:

- A function equivalent to an MSS stage-in operation. It makes a given file available on disk. It may recall a file from a local MSS but does not get a file from a remote site.
- A function equivalent to an MSS stage-out. It reserves space for a file on the Storage Element contacted.
- A Copy function that performs the copy a file between two SEs that can be both remote with respect to the client.

All calls above need to return a request token that can be used in the status request methods (see below).

- The PutDone function marks a new file as complete. At this point, the file can be copied to MSS.
- A function to extend the lifetime of a file to implement pinning capabilities. Several jobs/requests may give a different lifetime for a given file. The actual lifetime is the highest value.
- A function to unpin a file. The actual lifetime is recomputed from the remaining pin values.

- Functions to monitor the status of a given request: how many files are ready, how many files are in progress and how many are still queued.
- Suspend/resume/abort request functions.
- A function to negotiate file access protocols. The information system can publish the list of protocols supported by the storage service at a site. Also, the client connecting to the storage service can specify a list of protocols needed and negotiation can therefore start following the priorities dictated by the client.

Relative Paths
Experiments have expressed the need to refer to directory paths with respect to the VO base directory. This is to avoid finding out per site the root assigned by the site to a specific VO. The storage interface should discover the correct namespace for a given VO. Beisdes, in this way catalog entries could be shorter.

Other requirements
Although the storage interface under discussion is the strategic choice as the grid interface for all storage systems, the transfer systems are expected to also support other protocols (specifically GridFTP) at smaller sites.

The need to be able to send multiple requests to the MSS and to allow the MSS to handle priorities, and to optimize tape access was deemed as essential by the storage system managers, and recognized by the experiments. The use of the storage interface methods guarantees this ability, and would be the preferred method of initiating file transfers.

It is vital that all the storage interface implementations interoperate seamlessly with each other and appear the same to applications and grid services. To this end a test-suite has to be used to validate implementations against the WLCG agreed set of functionality and behavior.

3.5.3 The Storage Classes

In WLCG the Storage Class Working Group [78] has been established to understand the requirements of the LHC experiments in terms of quality of storage (Storage Classes) and the implementations of such requirements for the various storage solutions available. For instance, this implies to understand how to assign disk pools for LAN or WAN access and to try to devise common configurations for VOs and per site.

A ***Storage Class*** *determines the properties that a storage system needs to provide in order to store data.*

The LHC experiments have asked for the availability of combinations of the following storage devices: Tapes (or reliable storage system always referred to as tape in what follows) and Disks. If a file resides on Tape then we say that the file is in Tape1. If a file resides on an experiment-managed disk, we say that the file is in Disk1. Tape0 means that the file does not have a copy stored on a reliable storage system. Disk0 means that the disk where the copy of the file resides is managed by the system: if such a copy is not pinned or it is not being used, the system can delete it.

Following what has been decided in various WLCG Storage Class Working Group meetings and discussions only the following combinations (or Storage Classes) are needed and therefore supported:

- Custodial-Nearline: this is the so-called Tape1Disk0 class.

- Custodial-Online: this is the so-called Tape1Disk1 class
- Replica-Online: this is the so-called Tape0Disk1 class

- Tape0Disk0 is not implemented. It is pure scratch space that could be emulated using one of the available classes and removing the data explicitly once done. However, it could be handy for LHC VOs to have such a type of space actually implemented.

In the ***Custodial-Nearline*** storage class data is stored on some reliable secondary storage system (such as a robotic tape or DVD library). Access to data may imply certain latency. In WLCG this means that a copy of the file is on tape (Tape1). When a user accesses a file, the file is recalled in a cache that is managed by the system (Disk0). The file can be "*pinned*" for the time the application needs the file. However, the treatment of a pinned file on a system-managed disk is implementation dependent, some implementations choosing to honor pins and preventing additional requests, others removing unused on-line copies of files to make space for new requests.

In the ***Custodial-Online*** storage class data is always available on disk. A copy of the data resides permanently on tape, DVD or on a high-quality RAID system as well. The space owner (the virtual organization) manages the space available on disk. If no space is available in the disk area for a new file, the file creation operation fails. This storage class guarantees that a file is never removed by the system.

The ***Replica-Online*** storage class is implemented through the use of disk-based solutions not necessarily of high quality. The data resides on disk space managed by the virtual organization.

Through the Storage system interface, it is possible to schedule Storage Class Transitions for a list of files. Only the following transitions are allowed in WLCG:

- Tape1Disk1 -> Tape1Disk0. On some systems this can be implemented as a metadata operation only, while other systems may require more operations to guarantee such a transition.
- Tape1Disk0 -> Tape1Disk1. It was decided that this transition would be implemented with some restrictions: the request will complete successfully but the files will remain on tape. The files will be actually recalled from tape to disk only after an explicit request is executed. This is done in order to avoid that a big set of files is unnecessarily scheduled for staging and therefore to smoothen operations in particular for those Mass Storage Systems that do not have a scheduler (namely TSM).
- Tape0<->Tape1 transitions are not supported at the start of LHC (if ever). For physics validation operations, since the amount of data to transfer to tape after the validation is not big (only 1-2% of total data) a change class operation from Tape0Disk1 to Tape1DiskN can be approximated by copying the files to another part of the name space, specifying Tape1DiskN as the new storage class, and then removing the original entries.

Chapter 4

STORAGE SOLUTIONS

As we have seen, an application running on a Grid infrastructure needs to be able to transparently store and access data distributed at some storage centers. The Grid middleware is responsible for ensuring the transparency of operations over a multitude of hardware and software solutions used to provide reliable and performing storage. In this chapter we give an overview of the state of the art of Grid storage technologies. We first give as a motivating example the typical storage management operation of a WLCG computing center. In more detail, we present a summary and classification of commercial storage hardware solutions adopted in the WLCG infrastructure: the great variety of hardware solutions and products imposes the deployment of several management and data access software packages, which sometimes have non-negligible acquisition and maintenance costs. We also describe the software products developed by the various HEP laboratories in order to respond to specific needs and contain the costs. We outline the main features and limitations whenever possible.

4.1 A motivating example

Let us have a closer look at a typical situation in a small or medium-size WLCG computing center (a typical Tier-2 center). In general, small centers have only limited funding and resources. They might have a good relationship with some local vendors that provide them with good assistance and options in case of a hardware upgrade. In order to answer to the users' need for storage, a system administrator tends to use "anything" that is available. In these centers, the worker nodes of a farm can be equipped with a good amount of medium-quality disks or with small reliable storage systems. Instead of looking for some specialized storage solutions, the site administrators take full advantage of the hardware they already have available, and they look for solutions that make independent disk drives appear as one storage pool. Sometimes this is at the cost of management overhead. Therefore, in this case, directives coming from larger labs that have experience with the same products force the decision for the product to be chosen. To reduce the system administration overhead, other sites might decide to go for commercial solutions that can be easily adapted to the environment that the system administrator has to deal with. For instance, this was the case for many sites choosing to install and support classic distributed filesystems such as NFS or AFS. Some sites, even if small, have to satisfy the requests coming from a community of users who are really concerned with performance and stability issues. In these cases, it is very important to be able to monitor a system, balance the load, intervene immediately as soon as a problem arises, restore the system functionality, make the data available again to users as soon as possible, etc. All these requirements might become difficult to guarantee with storage solutions based on disks distributed on a farm of worker nodes. With the decrease of the costs of computing equipment, it is not infrequent today to find specialized disk or tape based storage technologies deployed at small computing centers. A good assistance provided by the vendor and the experiences of other centers make the managing task affordable. Figure 4.1 summarizes the solutions adopted today by WLCG Tier-2 centers.

Large centers such as the WLCG Tier-1 sites have stronger requirements. They must provide high capacity (tens of Petabytes), very reliable, and durable storage. Most of the times, this is achieved with tape-based mass storage systems. High performing tape drives are available in a tape library, a system that comes with a robotic arm and an optic label reader able to find the needed tape in the library and mount it in a free tape drive for reading and/or writing. Proprietary or custom written software drives the tape library. Some sites have decided to adapt proprietary backup or Hierarchical Storage Management (HSM)

systems to manage the robotic libraries and serve user requests. Some other sites have written custom solutions to do the job. Data are spooled on a set of disks, served by disk servers, before users read them. In write mode, data are rarely written directly to tape for optimization reasons. Disk buffers are used to store temporarily data that need to be migrated to tapes. Disk buffers are also used to transfer data outside the site and make them available to other centers.

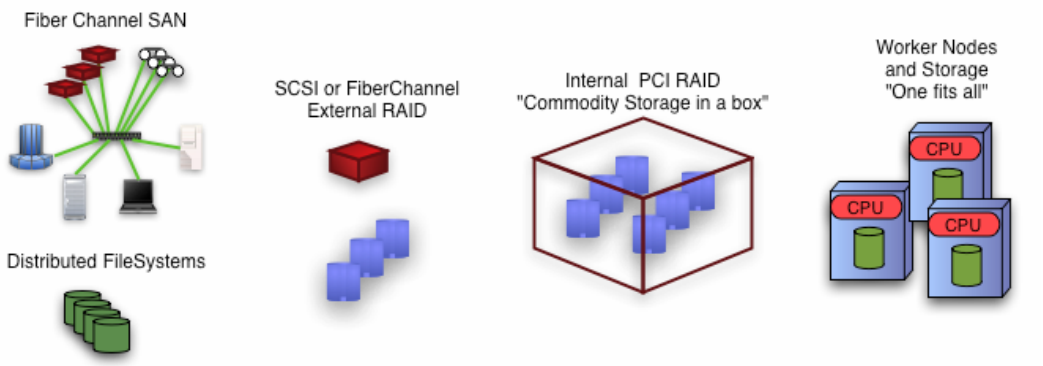


Figure 4.1. From complex expensive, systems [on the left] to inexpensive storage solutions [on the right].

In the next sections we describe hardware and software solutions for storage management, often called *storage appliances*.

4.2 Hardware storage technologies

In this section we describe the hardware technologies used to implement storage solutions. We distinguish between disk and tape based solutions. We point out limitations and advantages offered.

4.2.1 Disk based technologies

The most popular storage systems today used in small or medium size computing centers are small (~TB) disk-based storage products. They use low-cost parallel or serial **ATA disks** [90] and can operate at the block or file level. In addition, these disk-based products often use **RAID controllers** to offer reliable and performing capacity. RAID arrays or controllers are storage appliances that perform load balancing among self-contained storage modules.

ATA systems have been adopted to replace the old **direct-attached storage** (**DAS**) [90]: such old systems in fact present many problems. DAS uses a bus topology in which systems and storage are connected by a bus in a daisy chain (i.e. one device after the other). Data availability in such systems can be problematic since if any of the components fails, the entire system becomes unavailable. Other issues are limited scalability (only up to 15 devices in a chain), static configuration, captive resources and storage utilization, performance (only 40 MB/s), system administration, etc.

As an illustrative example, let us look at the solution adopted at CERN: about 450 ATA disk servers are used to serve about 650 TB of data using about 8000 disks spinning in "Commodity Storage in a box" cases. This solution is cost effective. However, the center adopting it must have a solid infrastructure in terms of monitoring, managing and intervening in case of problems. The number of disk servers and boxes that have to be manually managed could be quite high for the local support at a Tier-2. For

instance, at CERN a high failure rate of the disk drives has been noticed at the beginning of the setup. About 4% of the disks presented problems and had to be substituted. Furthermore, studies to guarantee the desired performance were conducted, and all servers have been reconfigured accordingly [92]. At CERN the "Commodity Storage in a box" has shown acceptable performance for the HEP environment. A typical configuration foresees an "xfs" Linux filesystem built on devices served by EIDE controllers configured with RAID5 (store of parity information but non redundant data) enabled at hardware level and RAID0 (striping) performed via software. RAID5 is a configuration for which parity information about data is stored on multiple disks in order to recover from data loss. RAID0 allow for writing/reading data in parallel to/from multiple disks so that performance increase with respect to read/write operations executed on a single disk. With such a configuration, write operations can achieve rates of 150 MB/s while in read mode data can flow at a rate of 250 MB/s.

Because "Commodity Storage in a box" comes with very little or almost no integrated management and monitoring tools and because of the very high system administration load that it imposes, many centers have chosen to invest in more reliable and manageable network storage appliances such as those built on Storage Area Networks (SAN) and the Network Attached Storage (NAS) solutions (cf. Section 4.3).

4.2.2 Tape based technologies

As of today, disk-based solutions can provide storage capacity to up to hundreds of Terabytes. In contrast, tape servers can provide Petabytes of storage capacity; however they often do not satisfy the performance requirements. Therefore, they are used essentially as tertiary data store accessible through a transparent user interface. Big robotic libraries are normally deployed in well-established computing facilities (cf. Figure 4.2). For example, at CERN 5 StorageTek silos are used to serve data or storage space on demand using specialized software to spool data on disk.

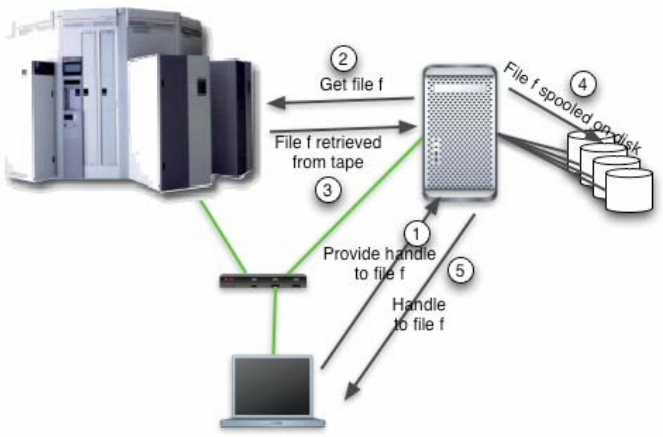


Figure 4.2. A user's request for a file located on a tape system. The figure shows the necessary steps that are required in order to serve the file to the user.

As an example, recently, using 45 installed StorageTek 9940B tape drives, capable of writing to tape at 30 MB/s, IT specialists at CERN were able to achieve storage-to-tape rates of 1.1 GB/s for periods of several hours, with peaks of 1.2 GB/s. The average sustained rate over a three-day period was of 920 MB/s.

Up to today, tape based systems have been considered to be a reliable and cost-effective solution for long-term and capacity storage. One of the main advantages is the power consumption of a tape-based

system per byte stored, which is much lower than for disk-based solution. However, the limitations of these systems are well known:

- Read/write rates are moderate (hundred of MB/s)
- Access to data is strictly sequential, which means that deleting data is a costly operation since tapes need to be "recompacted", i.e. the valid data need to be copied on new volumes to recover the space lost due to the deleted file.
- Mount/dismount operations are normally slow (in the order of seconds). Therefore, tape-access operations need to be prioritized and grouped in order to minimize mount/dismount operations.
- Tape technologies are rather complicated. Therefore, the error rate is normally higher than for disks.
- Different drives have different characteristics and most of the time proprietary drivers. Support for new drivers has to be constantly checked when performing a system upgrade.
- Constant maintenance needs to be foreseen for instance to clean the head of the tape drives and check performance and deterioration.
- The tapes themselves deteriorate with time. The data archived need to be recopied on new tapes sometimes using new types and even different formats, following the upgrade program at a site.
- Tape libraries are rather expensive, and the support provided by the vendor might not be adequate.
- As well as tape drives, also tape libraries need constant revision and maintenance.

4.3 Network based technologies

A ***Storage Area Network (SAN)*** [90] is a high-speed special-purpose network (or sub-network) that interconnects different kinds of data storage devices with associated data servers. They all communicate via a serial SCSI protocol (such as Fiber Channel or iSCSI). Storage devices participating in a SAN are therefore intelligent, and they can be addressed via an IP, appearing as separate nodes in a LAN. A SAN is usually clustered in close proximity to other computing resources but may also extend to remote locations for backup and archival storage.

A SAN can use existing communication technology such as IBM's optical fiber ESCON [91] or it may use the newer Fiber Channel [98] technology. SANs support the following features:

- Automatic disk mirroring;
- Backup and restore;
- Archiving and retrieval of archived data;
- Data migration from one storage device to another;
- Sharing of data among different servers in a network.

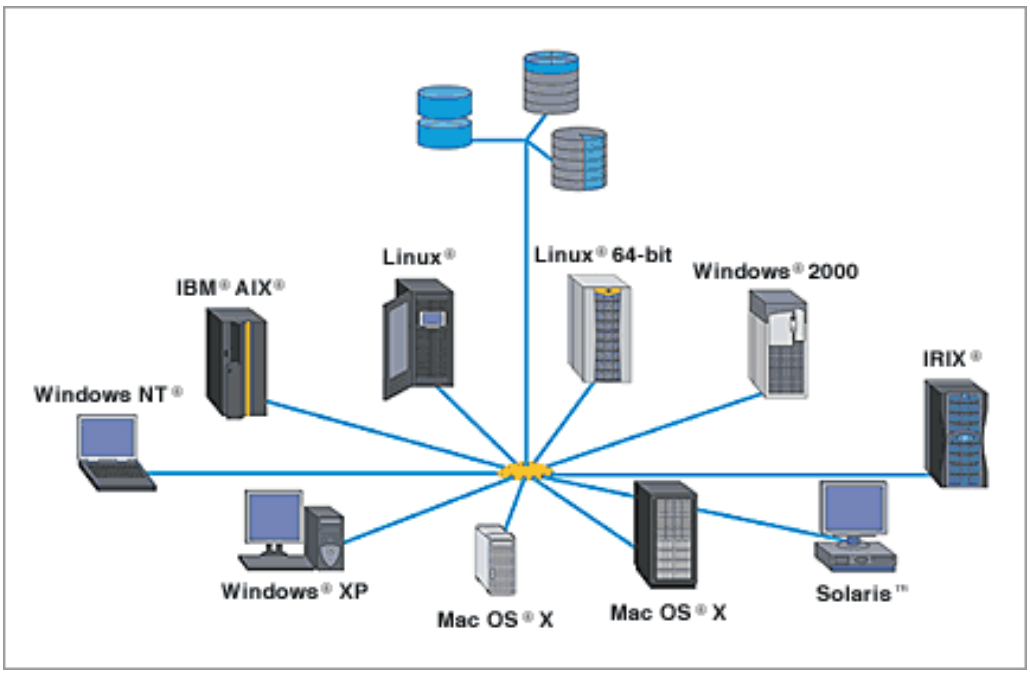


Figure 4.3. Example configuration of a SAN installation (source SGI).

SAN solutions operate at the block level. Therefore, they do not offer a namespace for the files stored in the storage devices participating in a SAN infrastructure.

While a SAN takes care of the "connectivity" and "storing" layers of a storage system, a ***Network Attached Storage (NAS)*** system takes care of "filing" operations. NAS [90] is a product concept that packages file system hardware and software with a complete storage I/O subsystem as an integrated file server solution. The network-attached storage device is attached to a local area network (typically, a Gigabit Ethernet) and assigned an IP address. The main server maps file requests to the NAS file server (or NAS head).

NAS servers have typically a few hundred GBs and extend up to tens of TBs of usable storage. They are normally specialized servers that can handle a number of network protocols, including Microsoft's Internetwork Packet Exchange, NetBEUI and CIFS, Novell's Netware Internetwork Packet Exchange, and Sun Microsystem's NFS. In particular, NAS heads are specialized machines in charge of the filesystem view of the data. Some NAS systems provide for dynamic load balancing capabilities, dynamic volume and file system expansion and offer a single, global namespace. NAS systems can deliver performance of tens of GB/s in a standard sequential read/write test.

However, besides being expensive systems, one of the problems with NAS systems is the incompatibility of proprietary solutions and the inexistence of interoperable NAS heads defining a global name space. A system administrator needs to independently manage different storage partitions defined by the different vendor products. For a comparison between NAS and SAN refer to Table 4.1.

NAS		SAN
Almost any machine that can connect to the LAN (or is interconnected to the LAN through a WAN) can use NFS, CIFS or HTTP protocol to connect to a NAS and share files.		Only server class devices with SCSI Fibre Channel can connect to the SAN. The Fibre Channel of the SAN has a limit of around 10km at best
A NAS identifies data by file name and byte offsets, transfers file data or file meta-data (file's owner, permissions, creation data, etc.), and handles security, user authentication, file locking		A SAN addresses data by disk block number and transfers raw disk blocks.
A NAS allows greater sharing of information especially between disparate operating systems such as Unix and NT.		File Sharing is operating system dependent and does not exist in many operating systems.
File System managed by NAS head unit		File System managed by servers
Backups and mirrors (utilizing features like NetApp's Snapshots) are done on files, not blocks, for a savings in bandwidth and time. A Snapshot can be tiny compared to its source volume.		Backups and mirrors require a block by block copy, even if blocks are empty. A mirror machine must be equal to or greater in capacity compared to the source volume.

Table 4.1. Comparison between the main SAN and NAS features (source NAS-SAN.com).

4.4 Software solutions

Let us now revise existing software systems that provide storage solutions on top of both disk and tape based storage hardware.

4.4.1 Disk Pool Managers (DPMs)

In the Grid community, there is a tendency to provide disk pool managers capable of serving large amounts of disk space distributed over several servers. However, most of the time, such systems do not allow for POSIX I/O, but file access is guaranteed via Grid or specific protocols. Furthermore, most of the time a Disk Pool Manager (DPM) presents the catalogue of available files to the users as a browseable file system-like tree; but a real file system is not available.

The development of such storage management software has started in the early 90s in order to provide second level storage for data stored in big robotic archiving systems. Through application software, a user asks for access to a file stored on tape. If the file is available on one of the second level storage disk servers, it is served either via custom remote I/O calls or copied on a disk directly accessible by the user. Otherwise, the robotic controller in the tape library is instructed to mount the tape containing a copy of the requested file on one of the available drives and it spools the file on the filesystem of one of the server managed by the DPM. The DPM is generally in charge of managing the disk space served by the storage servers. It is responsible for deleting from disk unused files saved on tape in order to make space for further requests, pinning a file on disk while it is used, reserving storage for new files, etc.

In what follows we give an overview of disk pool management systems deployed and used in production environment in the major HEP research labs.

4.4.1.1 dCache

dCache [64] is a software-only Grid storage appliance jointly developed by DESY and Fermilab. It is the DPM of the Enstore MSS. The file name space is represented in a single file system tree. dCache optimizes the throughput to and from data clients and smoothes the load of the connected disk storage nodes by dynamically replicating files. The system is tolerant against failures of its data servers. Access to data is provided by various FTP dialects, including GridFTP [14], as well as a proprietary protocol (gsi-dCap), offering POSIX-like file system operations like open/read, write, seek, stat, close. Some of the limitations of the dCache system are the complex configurability, some instability (shown during the CMS Data Challenge [99]) and the authorization control mechanisms. The Grid Security Infrastructure (GSI) [11] protocol implemented for file transfer allows for the check of user credentials in order to allow access to files. However, ACLs and mechanisms to enforce local access or priority policies are lacking.

4.4.1.2 LDPM

The LCG Lightweight Disk Pool Manager (LDPM) [100] is a complementary solution to the dCache system. It focuses on manageability and therefore it is easy to install and configure. It requires low effort for ongoing maintenance while allowing for easy addition and removal of resources. As for dCache, LDPM supports multiple physical partitions on several disk severs and allows for different types of disk space: volatile and permanent. In order to protect against disk failure LDPM provides support for multiple replicas of a file within the disk pools. As far as data access is concerned, also LDPM supports several proprietary POSIX-like protocols, such as rfio and ROOT I/O. GridFTP is used for file transfers outside the IP domain. The namespace is organized in a hierarchy.

Through plug-ins, several types of security mechanisms can be enforced: GSI, Kerberos, etc. Integration with the Virtual Organization Management Software (VOMS) [80] is also foreseen. Each user can have a specific role and privilege inside the Virtual Organization he is part of. The VOMS is responsible for describing the user role releasing appropriate credentials. The role of a Grid user is then mapped to LDPM Group IDs and enforced. The ownership of files is stored in the LDPM internal catalogues. UNIX and POSIX ACLs permissions are implemented. Recent developments foresee the integration of the LDPM with the Grid Policy Box middleware, G-PBox [102]. Such software allows for the enforcement of site policies and priorities (when a given user is allowed access to a site, with which priority, etc.). Plug-in policies are also foreseen for the pool selection strategy, the garbage collector, the request selection and the migration of files to other sites to prevent space exhaustion or for failure resilience. The LCG DPM targets Tier 1 and Tier 2 needs.

4.4.1.3 NeST

Another example of a flexibility software-only storage appliance is NeST [103]. Rather than focusing on a single protocol, NeST provides support for multiple data access and storage protocols such as HTTP, NFS, FTP, GridFTP etc. This is one of the strengths of NeST. Another focus is storage management and reservation comparable to SRM (see later). However, NeST does not manage large amounts of disk space but currently relies primarily on an underlying file system to provide access to data. NeST is Grid aware in a sense that it uses Grid security as well as provides for GridFTP support in order to transfer data to and from NeST within a Grid environment.

4.4.1.4 DRM

A Disk Resource Manager (DRM) [104] has been developed by LBL and can be regarded as one of the predecessors of the SRM. In more detail, a disk cache is available through the operating system that provides a filesystem view of the disk cache, with the usual capability to create and delete directories/files, and to open, read, write, and close files. However, space is not pre-allocated to clients. Rather, the amount of space allocated to each client is managed dynamically by the DRM. In its current implementation DRM is based on CORBA and provides an SRM interface.

4.4.1.5 SAM

The Storage Access Manager (SAM) [105] developed at Fermilab tries to tackle the storage problem in a slightly different way. The storage resource is considered one of the possible Grid resources and as such is one of the variables in the scheduling decisions for a job. SAM is therefore a job management system with data management integrated patterns. It is interfaced with Enstore [101] and dCache. Before scheduling a job, SAM makes sure that the data sets needed by the application are available at the site where the computation happens. In case they are not, SAM schedules data retrieval and data transfer to the computing site where the job is scheduled to run. SAM is a completely integrated solution and it is "hard" to interface to other solutions or Grid infrastructures, such as LCG. Therefore, at the moment, it has been adopted only at FNAL and at collaborating sites.

4.4.2 Grid Storage

Lately, the term "Grid storage" has crept into the product literature of vendors and refers to two items: a topology for scaling the capacity of NAS in response to application requirements, and a technology for enabling and managing a single file system so that it can span an increasing volume of storage. Scaling horizontally means adding more NAS arrays to a LAN. This works until the number of NAS machines becomes unmanageable. In a "Grid" topology, NAS heads are joined together using clustering technology to create one virtual head. NAS heads are the components containing a thin operating system optimized for NFS (or proprietary) protocol support and storage device attachment. Conversely, the vertical scaling of NAS is accomplished by adding more disk drives to an array. Scalability is affected by NAS file system addressing limits (how many file names you can read and write) and by physical features such as the bandwidth of the interconnect between the NAS head and the back-end disk. In general, the more disks placed behind a NAS head, the greater the likelihood the system will become inefficient because of concentrated load or interconnect saturation. Grid storage, in theory, attacks these limits by joining NAS heads into highly scalable clusters and by alleviating the constraints of file system address space through the use of an extensible file system. The development of storage Grids clearly is geared toward NAS users today, but others might one day benefit from the Grid storage concept. For instance, making disparate SANs communicate and share data with each other in the face of non-interoperable switching equipment is today a complicated problem to solve. By using clustered NAS devices serving as gateways and managers of the back-end SANs, one would gain improved capacity, file sharing and management.

At IBM's Almaden Research Center, work is proceeding on a Grid storage project aimed at creating a "wide-area files sharing" approach. In the Distributed Storage Tank (DST) project, the objective is to extend the capabilities in a "Storage Tank" - a set of storage technologies IBM offers that includes virtualization services, file services and centralized management - to meet the needs of large, geographically distributed corporations. IBM is looking at not yet used capabilities in the NFS Version 4 standard to help meet the need. DST extends to NFS clusters that can be used to build a much larger

Grid with a single global file namespace across a geographically distributed and heterogeneous environment. Making the approach open and standards-based requires a schema for file sharing that is independent of a server's file and operating systems, and that does not require the deployment of a proprietary client on all machines. IBM is working with the Open Grid Forum's File System Working Group [114] because its intent is to produce a standards-based Lightweight Directory Access Protocol (LDAP) server to act as the master namespace server.

4.4.3 Distributed File Systems

From a user point of few, a file system is the most "natural" storage system to use since it is typically integrates into the operating system. However, conventional UNIX file systems are designed to manage a single or only a small number of disks. In case many disks should be connected, distributed file systems are needed. These are sometimes also called "cluster file systems".

These file systems are an alternative form of shared file system technology. They do not use a separate meta-data server but are designed to work only in homogeneous server environments where improving storage manageability is not a goal. However, they are used at many centers as a solution to share storage among a farm of computing nodes. Using very high-speed connections (Switched Gigabit Ethernet, Infiniband, etc.) such solutions provide for POSIX I/O, centralized management, load balancing, monitoring, and fail-over capabilities, among others.

File systems such as **NFS** and **AFS** are widely used, but they present quite a few performance and scalability problems [112,113]. NFS with its current implementation of protocol version is mainly used for local area networks whereas AFS can be used as a worldwide filesystem spanning sites connected via wide area networks.

Another but not so commonly used distributed, network file system is SGI's **CXFS** [109]. It separates the data and metadata servers from each other. Furthermore, CXFS provides direct access to SAN. Another significant difference is the way file locking is implemented: it is achieved via a metadata broker rather than by individual hosts.

4.4.4 Parallel File Systems

Similar to distributed file system, also parallel file systems can be used on clusters managed via a LAN. However, by providing parallel access to data via several data servers, parallel file systems overcome the performance bottleneck experienced with conventional distributed file systems.

The following motivations suggested the use of parallel file systems in the Grid. In particular, users do not always have full control over their applications. Adopted proprietary solutions, legacy software, performance factors, etc. often do not allow for changing (re-writing) an application in order to make it Grid-aware. The integration of existing high-performance, parallel file-systems into a Grid infrastructure allows users to take advantage of such technologies. Widely used, high-performance distributed file-systems are IBM/GPFS [115], Lustre [116], and PVFS-2 [111].

4.4.4.1 GPFS

The IBM General Parallel File System (GPFS) for Linux is a high-performance shared-disk file-system that can provide data access from all nodes in a Linux cluster environment. Parallel and serial applications can access shared files using standard UNIX file-system interfaces, and the same file can be accessed concurrently from multiple nodes. GPFS provides high availability through logging and replication, and can be configured for fail-over from both disk and server malfunctions. To support its performance objectives, GPFS is implemented using data striping across multiple disks and multiple nodes, and it employs client-side data caching. GPFS provides large block size options for highly efficient I/O and has the ability to perform read-ahead and write-behind file functions. GPFS uses block level locking based on a sophisticated token management system designed to provide data consistency while allowing multiple application nodes concurrent access to a file. When hardware resource demands are high, GPFS can find an available path to the data by using multiple, independent paths to the same file data from anywhere in the cluster, when the underlying storage infrastructure allows it.

4.4.4.2 LUSTRE

LUSTRE has features similar to those of GPFS. It is a commercial product by Cluster File System, Inc. initially distributed free of charge. It is advertised as scalable to more than 10,000 clients. It is stable and reliable but quite invasive in terms of changes to the system kernel. It offers metadata redundancy and multiple metadata servers for fault tolerance. Only plain striping of data across the servers is allowed at the moment. Data recovery features and a POSIX interface are also offered.

4.4.4.3 PVFS

The Parallel Virtual File System (PVFS) project is conducted jointly between The Parallel Architecture Research Laboratory at Clemson University and The Mathematics and Computer Science Division at Argonne National Laboratory. PVFS is “easy” to install and “very light" (it can use the underlying native file system to store data), and provides user-controlled striping of files across nodes. Beside standard UNIX I/O, PVFS provides multiple I/O interfaces, such as MPI-IO via ROMIO.

4.4.4.4 StoRM

The StoRM software package does not provide a proper parallel file system. However, it implements a Grid storage solution based on parallel or distributed file systems and therefore we list it in this section. StoRM is the result of a research collaboration between INFN and ICTP (International Centre for Theoretical Physics, in Trieste, Italy) to build a disk-based SRM service on top of high-performance parallel file systems and at the same time provide a pilot national Grid facility for research in economics and finance. StoRM is designed to support guaranteed space reservation, direct file access via native POSIX I/O calls, and a solid security framework based on VOMS X.509 extended certificates.

A modular architecture (cf. Figure 4.4) decouples the StoRM core logic from the underlying file systems. Therefore, StoRM can support any file system that complies with the XDSM Specification [118] for space (disk block) allocation. The current implementation of StoRM supports IBM GPFS and SGI CXFS. StoRM also provides support for ACLs if the underlying file system complies with the POSIX 1003.1e/1003.2c draft 17 [117]. ACL entries are created on physical files for the local user corresponding to the Grid credentials associated with the VOMS proxy.

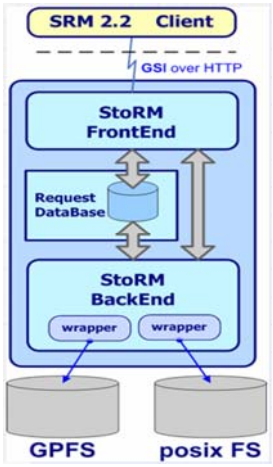


Figure 4.4 The StoRM Architecture

For what concerns enforcement of local policies, StoRM can be configured to use an external policy decision point (PDP). For instance, when a user requests access to a file stored in StoRM, the system verifies if the user holds a valid proxy certificate. Then, it contacts the PDP to verify if the user is authorized to perform the specific action. If the user is authorized, then the mapping service (LCMAPS) is contacted to retrieve the local account corresponding to the Grid identity of the requester. The StoRM file system wrapper is then invoked to enforce permissions by setting a new ACL on the physical file. The user can proceed to access the file via standard POSIX I/O calls. Static ACLs can also be enforced at group level.

The gLite G-PBOX service is queried by StoRM to check for policies and priorities established at site level for the specific Storage Element.

4.4.5 Mass Storage Systems

If disk space is not sufficient, a Mass Storage System (MSS) combines both secondary storage (disks) and tertiary storage (tape systems) and provides one uniform interface. Here we list a few of the most popular MSS mainly used in the HEP community.

Among the most commonly used MSS software products used to manage tape based solutions are: CASTOR [65] developed at CERN, ENSTORE [101] developed jointly by FermiLab, near Chicago, and DESY in Germany.

4.4.5.1 CASTOR

CASTOR, the Cern Advanced STORage system, is a scalable, high throughput storage management system. The CASTOR architecture is shown in Figure 4.5.

The main client interaction is done via the RFIO protocol (cf. Chapter 5 for further details). One of the most important components of CASTOR is the stager, which is responsible for disk cache management. The name server provides a standard file system hierarchy.

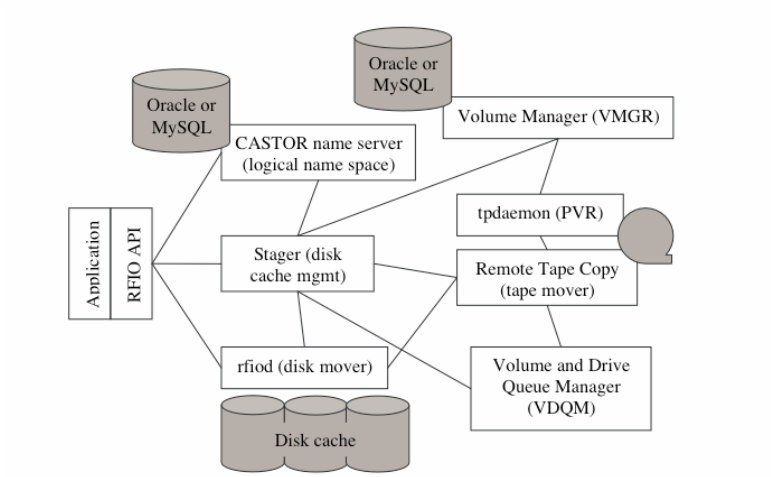


Figure 4.5. The CASTOR architecture.

4.4.5.2 HPSS

Another MSS is the High Performance Storage System (HPSS) [106] that started as a joint effort between industries and research. It has been developed to provide an answer to the need coming from the research field of providing automatic and transparent access to tape storage. The commercial market normally provides only backup solutions and Hierarchical Storage Manager (HSM) software, which often do not satisfy the requirements of fast access to data mainly in read-only mode.

4.4.5.3 TSM

TSM (Tivoli Storage Manager) [107] is mainly a backup and recovery management system. It is IBM's solution to storage management and it allows for transparent access to tape libraries. TSM is used at FZK in Karlsruhe (Germany) as a tape backend for dCache.

4.4.5.4 DMF

DMF (Data Migration Facility) [108] is an SGI Hierarchical Storage Manager (HSM) that allows for transparent access to data stored on tape. DMF foresees a set of disk servers in front of tape libraries to act as local cache for files that are stored or recalled to/from tape. Site administrators can define policies for cache management. For instance, new files are automatically scheduled for being copied to tape. The service that migrates files to tape in order to free up the space in the local cache runs at a scheduled time. It starts the migration of scheduled files to tapes only when the disk is full for a percentage threshold known as "high-level water mark". Files are migrated to tape and/or deleted from the disk cache until the "low-level water mark" is reached, i.e. the disk cache has reached a certain defined percentage of free space. A policy can be defined to have the system choose among all possible files candidate to migration. When a user requests a file, this is automatically retrieved from tape and put on disk where the user can directly access it. The file namespace is organized as for a UNIX filesystem. DMF has been integrated at SARA to work as a backend for the dCache system, in order to optimize file access and make the DMF space available on the Grid.

4.4.5.5 SRB

The Storage Resource Broker (SRB) [110] developed at San Diego Super Computing Center is client-server middleware that provides uniform access for connecting to heterogeneous data resources over a wide-area network and accessing replicated data sets. It uses a centralized Meta Data Catalog (MCat) and supports archiving, caching, synchs and backups, third-party copy and move, version control, locking, pinning, aggregated data movement and a Global Name space (filesystem like browsing). SRB provides as well for collection and data abstraction presenting a Web Service interface. The SRB has been integrated in the LCG Grid infrastructure; however, the centralized SRB catalogue has shown its limitations.

FILE ACCESS AND TRANSFER PROTOCOLS

In the HEP environment there is a very high demand for supporting remote file access I/O using existing and very well established remote file access protocols, such as rfio, dcap, and xrootd, but also for very performing data transfer protocols, adapted to the Grid environment.

In the previous chapter we introduced the various hardware and software components to realize Grid storage systems. We already mentioned the need for remote file access methods to (mass) storage systems both in LAN and WAN domains in Section 3.4. Furthermore, in chapter 6 we have highlighted the need for efficient and performing data transfer protocols. We now look at the details of such data transfer and file access protocols provided by particular storage systems. We also outline their functionalities and limitations wherever possible.

5.1 Data Transfer Protocols: GridFTP

The GridFTP is part of the family of protocols and tools introduced by the Globus project to establish a Grid environment. GridFTP is a command/response protocol. A client sends a command to the server and then it accepts responses from the server until it receives the one that indicates that the server is finished with that command. GridFTP uses two channels. The control channel is used for sending commands and responses. It is of low bandwidth, and is encrypted for security reasons. The second channel is known as the data channel. Its sole purpose is to transfer the data. It is of high bandwidth and uses an efficient protocol. By default, the data channel is authenticated at connection time, but no integrity checking or encryption is done due to performance reasons. Integrity checking and encryption are both available via the client and libraries. A ***network endpoint*** *is a point of access to the network with an associated IP address (a network interface card) for transmission and reception of data*. In GridFTP the data channel may actually consist of several TCP streams from multiple hosts. GridFTP allows in fact for ***parallelism***, i.e. *for establishing multiple TCP connections between a single pair of network endpoints*. This is used to improve performance of transfers on connections with light to moderate packet loss. With a cluster of nodes sharing the same file systems it is convenient also to use striping. ***Striping*** *refers to having multiple network endpoints at the source, destination, or both, participating in the transfer of the same file*. GridFTP allows also for concurrency. ***Concurrency*** *refers to having multiple files in transit at the same time*. They may all be on the same host or across multiple hosts.

The first release (version 1) of the GridFTP protocol presented already many features that are fundamental in a Grid environment. Among the features that extend the functionalities of a classical FTPd server, we list the following:

- GridFTP is ***GSI enabled***. It offers authentication, integrity and confidentiality on the control channel, besides allowing for customization and enforcement of local policies. GridFTP offers as well Kerberos authentication, with user-controlled setting of various levels of data integrity and/or confidentiality. Kerberos, however, does not support delegation. Therefore data channel authentication is not possible under standard Kerberos.
- GridFTP allows for ***third-party data transfer***. This capability is already defined by the FTP standard. In fact, control and data channel can exists on different hosts on the WAN. The client

establishes two control channels, one to each server and listens on one of them. It then sends the IP/port pair of one of the server to the other which then connects to initiate the actual movement of the data. However, GSSAPI security has been build on top of the existing third-party controlled transfer capability.

- ***Multiple TCP streams*** can sometime improve aggregate bandwidth on a wide area network. Therefore, GridFTP supports parallel data transfer through FTP command extensions and data channel extensions.
- GridFTP supports as well ***striped data transfers*** over multiple servers through extensions defined in the Global Grid Forum draft.
- Another extension concerns full ***partial data transfer support***, a feature requested by many applications. The standard allows copying the remainder of a file starting at a certain offset. GridFTP introduces the concept of a region of the file and allows for its transfer.
- ***Reliable data transfer*** extensions have been added to the protocol to make it more robust and allow for recovering of failed session.
- In v1, the protocol has support for manual TCP/IP buffer size tuning, a critical parameter to reach optimal performance.

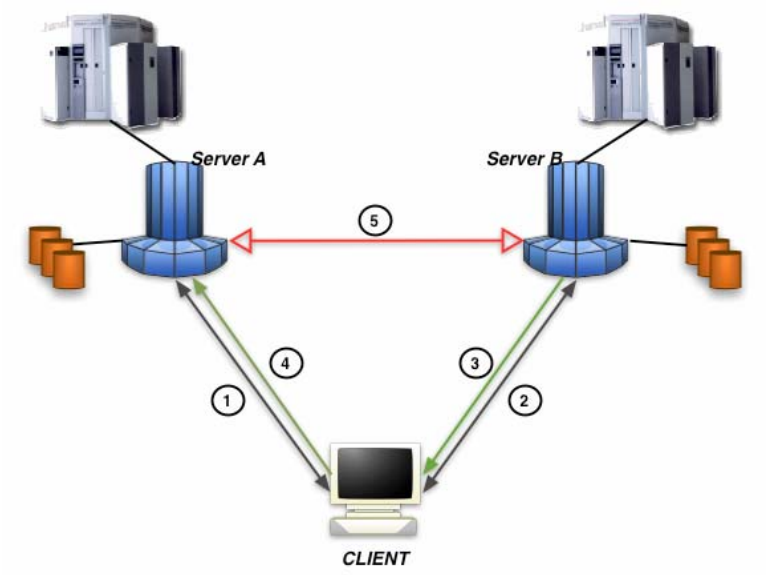


Figure 5.1 Third-party transfer: Client and server authenticate in steps 1,2. Server B sends data channel IP/Port to client (step 3). Client sends Server B IP/Port to Server A (step 4). Data transfer starts (step 5).

The WLCG middleware is about to migrate to version 2 of the protocol that has many advantages with respect to version 1. One of the limitations of version 1 was the need for the server to establish an active connection with the client when parallel data transfer was chosen. In fact, version 1 of the GridFTP protocol allows for the dynamic creation of additional communication sockets. This feature introduces a potential race condition with an in-flight socket connection and end-of-file processing. To avoid this problem, the protocol required that the host that is sending the data performs the TCP connect. The resulting directionality causes problems with firewalls since active connections from an outside server are normally denied.

Version 2 of the GridFTP protocol overcomes this limitation. The X (Extended) Mode of operation has been introduced and used to overcome many of the limitations of version 1. Among the features of version 2 we list the following:

- Dynamic data channel management. TCP/IP buffer (window size) tuning is now performed automatically.
- Data integrity verification with an option to retransmit corrupted blocks during data transfer. This includes the possibility of resending corrupted or missing blocks, and inclusion of a transfer ID to allow pipelining and de-multiplexing of commands.
- Data integrity commands have been introduced to send the file checksum ahead of the file to be transferred and to get the checksum for a file or a portion of it.
- Data flow independent of data socket connection. This allows even clients behind a firewall or on a NAT to use multiple streams. In fact, support for striping is introduced directly in the protocol that now allows for requesting a set of data channels for data transfer instead of just one.
- The GridFTP protocol defines "restart markers," messages sent to the client over the control channel by the writing server to indicate that a block has been successfully written to disk. If a remote failure occurs, be it network, remote host, server, etc., the client can restart the transfer and provide the restart markers. The transfer picks up from where it left off. Note that if the client fails, the restart markers are lost since they are held in RAM. If greater reliability is needed, this must be provided by higher-level services such as the WLCG File Transfer Service (FTS) [100].
- Data channel is established only after the server knows about the file to transfer and other transfer parameters. This allows for optimization and selection of transfer buffers, especially in case of hierarchical storage manager.
- If the client disconnects without sending and EndOfFile (EOF) or Bye command, the last transfer is considered corrupted and therefore it is discarded. This allows hierarchical storage managers not to store corrupted data transfer on tape.
- Concurrent transfer of multiple files in the same transfer session. Individual transfers are identified by a transaction ID that is sent with each X-mode block. Each block carries as well the offset and length of the data with it.
- The extended retrieve (ERET) and store (ESTO) commands have been modified to allow users to invoke custom algorithms that have been added to the server. The negotiation mechanism allows a client to determine which algorithms are available.
- Security is a major concern in Grid applications. FTP provided no security on the data channel. The data channel authentication (DCAU) command has been added to allow for authentication, integrity checking, and encryption of the data channel.

GridFTP is certainly a basic Grid protocol and the Globus implementation of it is in wide use by many Grid projects. An implementation of version 2 of the protocol is available and distributed with the Globus Toolkit GT4. The Open Grid Forum has specified the Recommended Standard for GridFTP in GFD.020.

5.2 File Access Protocols

Let us start with a basic use case. We assume that an LHC physicist needs to access reconstructed data stored on tape and managed by CASTOR. In addition, there is a Grid interface in front of CASTOR to allow for staging and pinning files, as we will see in Chapter 6. Let us suppose that the end user wants to use the CASTOR specific access protocol RFIO (details given below) to read data from the file. The Grid

interface will allow for this possibility. However, the physicist needs to explicitly specify which access protocol his application will use (in our case RFIO). The storage Grid interface then returns a handle to the requested file using the specified protocol. In our example, the Tranfer URL (TURL, as defined in Chapter 2) looks as follows:

```
rfio://hostname.org:4556/data/reco/file.root
```

This TURL can then be used with the RFIO specific calls for open and read to access the data.

In the next three sub sections we discuss three different access protocols for the following for storage systems:

File access protocol	Supported storage systems
RFIO	CASTOR, HPSS
Dcap	dCache
Xrootd	ROOT, others

At the end of this chapter we make some considerations about the importance of providing support for standard POSIX I/O and in particular for very performing filesystems such as the parallel ones. This request comes especially from science communities such as the Bioinformatics groups using legacy or proprietary software not modifiable to allow for integration with the Grid.

5.2.1 RFIO

RFIO (Remote File Input/Output) allows for access to non-local files mainly stored in CASTOR and HPSS [65]. In addition to providing a protocol, RFIO is a full client-server system that provides both a client library as well as server called **rfiod**.

RFIO provides both a command line interface and a language API in C and C++. The basic command line interface is as follows and the individual tools correspond to the equivalent UNIX file system command line tools:

```
rfcat, rfchmod, rfcp, rfdir, rfmkdir, rfrename, rfrm, rfstat
```

The C API is very similar to the POSIX file system interface for accessing files. For instance, the system call

```
int open(const char *, int, ...)
```

corresponds to the following RFIO function :

```
int rfio_open(const char *, int, ...)
```

The main difference between the RFIO command and its equivalent in POSIX is that besides operating on a local file, RFIO can also manage a remote file. Such file could be managed as well by a mass storage system such as CASTOR. In general, the most common POSIX file access methods such as read, fseek, write, close, etc. are redefined in RFIO to operate on remote files. This should make it easy to port existing applications to RFIO. RFIO is particularly optimized to work on a LAN. TCP buffer sizes can be automatically negotiated between client and server but can also be manually configured for better performance. Also, local policies to allow or deny access to certain hosts can be enforced.

Authentication was originally performed in rfio using UNIX uid/gid mapping. A user had to have an account on the machine where the rfio server was running in order to have access to files served by that machine.

In order to make it suitable for a Grid environment, rfio has been enhanced integrating GSI authentication and authorization. The usual technologies used for other Grid services in WLCG are used as well for rfio. After the verification of the Grid X.509 proxy that can contain VOMS extensions for roles and groups definition, the LCMAP service is invoked to check authorization and therefore map the user requesting the service to a local account enable to the specific privileges requested. Authorization and authentication are performed once at connection time, while data travel unencrypted between client and server for performance reasons. Data encryption can also be enabled but it is not supported at the moment.

In addition to the C interface there is also a C++ interface to RFIO. This basically extends the C++ stream class to operate on remote files.

5.2.2 dCap

dCap (Disk Cache Access Protocol) is the dCache native access protocol [64]. It provides POSIX-like functions such as `open`, `read`, `write`, `create` and `lseek` into the dCache storage. Additional administrative and debugging tools are available, too.

All interactions via dCap are done in a secure way. Three different security modes are supported: plain text, GSI and Kerberos. Similar to RFIO, dCap provides both command line tools and a POSIX like C API. The command line tools are as follows:

- `dccp` for copying data into and out of dCache.
- `dc_stage` for prestaging a file out of dCache
- `dc_check` checks if a file is on disk in dCache

The C API functions are very similar to their equivalents in the POSIX API. For example, an open call is realized as follows:

```
int dc_open(const char *path, int oflag, /* mode_t mode */...)
```

The number of API methods supported by dCap is not as large as the one provided by RFIO but from the point of view of functionality they are quite similar. In addition to the basic POSIX calls there are separate calls for file buffering (read ahead buffer) and fast, unsafe writing of files via callback operations for dealing with firewalls. Furthermore, TCP buffer sizes can be tuned for efficient data transfer.

A typical dCap TURL looks like follows:

```
dcap://<host>:<port>/</pnfs>/<storage_group>/usr/<filePath>
```

where host and port are the hostname and port where the dcap daemon is running, </pnfs> is the root of the dCache filesystem namespace, <storage_group> defines the dCache storage quality (tapes, disks, etc), and <filePath> is finally the path of the file.

5.2.3 xrootd

Whereas RFIO and dCap provide libraries and access to specific storage systems such as CASTOR and dCap, xrootd is a storage system independent data access protocol. In addition, the xrootd system (also referred to as Scalla [85]) provides a low latency, high bandwidth data access solution, i.e. the xrootd system itself provides a scalable storage solution. Access to the storage is achieved via the xrootd protocol, that provides basic POSIX like access to files as well as a powerful redirection mechanism to access high availability clusters.

Internally, the xrootd system consists of one or more xrootd data servers (simply called xrootd data server) as well as olb (open load balancing daemon) servers that are responsible for clustering of servers, providing a unique namespace across multiple data servers and load balancing data access requests. A simple schematic view can be seen in Figure X.

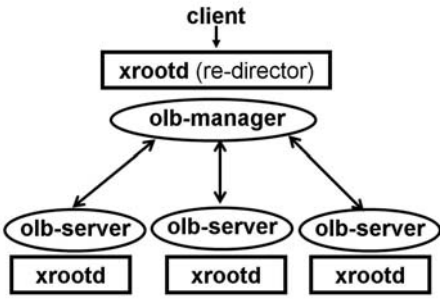


Figure 5.2. The xrootd system with data servers and load balancers. Clients only know about data servers (xrootd) and do not know about any of the olb servers that are used for redirection.

The main difference between the previously introduced protocols and xrootd is the following one:

- RFIO, dCap: A client connects to a data server, opens a file and then reads or writes from/to it. The data server is "passive" in a way that it returns data to the client as requested without taking any further actions. In case a data server gets overloaded or cannot handle additional requests anymore, it is up to the client to find another suitable data server.
- xrootd: Rather than keeping a connection to the data server for the whole time needed by the request, the xrootd protocol allows for redirection at any time in order to redirect the client to another server either to provide a better service or to balance the load between different data servers. This is a fundamental difference in the way open, read and write operations are perceived by the client: the server can decide at any point in time to stop the connection and refer that client to another data server. The xrootd protocol handles all the necessary interactions for achieving this and restarting open, read or write operations on another data server.

The xrootd protocol is an open protocol that is based on the request-response paradigm. Protocol exchange between a client and a server is done in a way that all numeric values are transferred in network format (i.e. big endian) and then have to be transferred to the architecture type of the receiver. For a detailed explanation of the xrootd protocol refer to [86].

Xrootd has been developed during the time when the Grid efforts in the area of storage management and access started. However, the approach was different and addressed mainly to optimized and fast disk file access. In particular, the interface to tape-based mass storage systems has been implemented through

plug-ins not following a specific standard. Therefore, recent efforts have tried to integrate the features and attractive functionalities of the xrootd (such as the automatic redirections) with the proposed interfaces and control protocols for storage access and management.

Chapter 6

THE STORAGE RESOURCE MANAGER INTERFACE

In this chapter we concentrate on Grid storage interfaces and protocols for managing and accessing space in a Grid environment and working with data stored in Grid storage systems. We present the motivations that have brought researchers to concentrate on this issue and give a brief history of the work done by Grid communities to tackle storage access and management in the Grid. In particular, we focus on the Storage Resource Manager interface for standardizing the access to storage and defining a set of needed features.

6.1 Motivations and History

As we have already outlined in previous chapters, at each computing center participating to WLCG, storage solutions are realized using:

- Commercial solutions that bundle together proprietary specialized hardware and software
- Commodity storage products that offer modest performance (see Chapter 4)
- Software based storage appliances that can make multipurpose machines act as storage servers for disperse disk storage devices.

In what follows we outline the motivations that have brought researchers to define a storage resource manager protocol.

6.1.1 Motivations and requirements

The need for reliable, scalable, manageable and high performing network storage has led to a proliferation of commercial and customized solutions. Each site makes its own choice for a storage system depending on many factors: particularly positive experience and good contacts with some vendors, budget considerations, local manpower to maintain and administer the system, special contacts with reference sites that give guidelines on solutions to adopt, etc. As a result, in the Grid we have a great deal of products deployed, all presenting good features and performance, but with a variety of interfaces and implementations. Furthermore, such solutions were designed either for wide area or local use: none of those designs seems guided by practical considerations about the nature of the Grid such as the following:

1. Grid users and applications migrate across multiple administrative domains. This has numerous diverse consequences: both a ***transparent interface*** for specific, very frequent operations and a ***set of different communication protocols*** need to be supported at the same time. Also and more important, ***security protocols*** used across domains might differ and must be equally and transparently supported. In particular, authentication and authorization mechanisms valid across domains must be honored.
2. Efficient methods for both local and wide area access must be provided. Reliable and high performing ***data transfers protocols*** supported across domains on wide area networks are fundamental.

3. Even though Grid-wide privileges should be guaranteed, ***local policies and priorities*** must be respected.
4. Beside the normal user who needs to store permanently his/her data, the Grid introduces a new set of users, the migratory users who use the local storage as intermediate storage to execute transient operations. Therefore a set of ***storage classes*** must be available and published in the information system so that application and services can take advantage of them. A storage class (see Section 3.5.3) is a type of storage space with specific attributes. For instance it might offer reliable, persistent storage or unreliable scratch space for temporary usage only.
5. Grid users also need to have available a set of operations for ***managing and reserving storage space*** and for ***filesystem-like operations*** (such as ls, mkdir, ln, file locks, etc.) as well as different ***qualities of services*** that need to be supported by the local storage solution.
6. Storage services should also be able to differentiate between valuable and expendable data (***volatile*** vs. ***permanent data***) when expired reservations are selected for removal. Operations such as ***transparent***, ***automatic*** or ***forced migration to tertiary storage*** (tapes) should be available. In order to avoid that a file in use is removed by a concurrent application, a mechanism called ***pinning*** is used to keep the system from removing files.
7. Mechanisms for transparently ***locating data*** on any storage device must be provided for debugging reasons and for recovering from disasters, besides implementing specific application needs.
8. Storage systems should also provide a mechanism to ***advertise capacity, status, availability and content*** to an information system.
9. ***Management*** and ***monitoring functions*** for Grid global control of service behavior and functionality are important.
10. ***Support for multiple file access and discovery protocols*** is requested for those (legacy) applications that need remote file location and access***.***

Therefore, it was felt by the community of Grid researchers that a big effort was needed in order to understand the needs and requirements for a Grid storage service and therefore to start the process for the definition of a storage service model and standard storage management and access protocols. The definition of an abstract storage interface would have allowed for the virtualization of the storage resources and for a wide range of physical storage devices to be optimally utilized.

6.1.2 History and related work

The idea of a wide area network storage service is not new and there are many products that supply such a service since many years. In Chapter 4 we have described the functionalities and capabilities of products such as the Storage Resource Broker (SRB) or HPSS. These products address many of the issues common to a Grid environment, however they do not propose a standard and flexible approach for accommodating the heterogeneity of the Grid.

Real discussions about the need of common protocols started around the end of the 90s with the promotion of Grid-like middleware such as Condor and Legion (see Chapter 2). At Super Computing 1999 the Globus Project started the discussion about Grid storage protocols demonstrating a prototype called "*GridStorage*". The main outcome of this presentation was to stress the need of, and provide users with a prototype for, a very reliable protocol for file transfers. In May 2000 the first implementation of the GridFTP protocol was done modifying the Washington University version of the FTP daemon (wuftpd) and making it GSI aware. In 2002, the GridFTP working group became a formal group of the

OGF. However the focus of this working group was on efficient and reliable data transfer at a file (or partial file) level.

In 1998, within the Global Distributed Network Storage project promoted by the University of Tennessee, Knoxville, the Internet Backplane Protocol (IBP) [87] was proposed. IBP is an unreliable network block storage protocol that offers storage allocation, read/write and storage management operations over the network. However, this protocol addresses disk storage resources only and imposes internal file access and transfer protocols in order to strictly control the bytes written on the storage resource. Therefore, it is difficult to adapt legacy applications that run in a Grid environment to use IBP. Furthermore, IBP decouples user identification from storage access, allowing for anonymous access to subsets of resources available on the wide area network. This together with the other issues mentioned made the IBP not really attractive for the Grid community. The IBP is surely a precursor of the SRM interface as discussed today in the OGF.

The concept of SRM was introduced as a result of an early project funded by Department of Energy (DoE) at LBNL in 2001. This was based on the ideas and experience with the STACS system [119] that provided a cache manager to stage files from an HPSS system in order to automatically serve them to clients. However, it was only in June 2003 (GGF-Bird Of a Feather) that an OGF working group (SRM-WG) focusing on storage interfaces and storage resource management was promoted by LBNL in conjunction with Jefferson Lab (JLAB), Fermi National Accelerator Laboratory (FNAL), and CERN. The SRM-WG used a few implementations, mainly disk-based, in order to prove the validity of the proposed SRM interface. The idea was to start from an FTP-like interface and extend it in order to accommodate other requests for space reservation and file pinning. When the EGEE project started (April 2004) it was clear that the version proposed for deployment of the protocol just defined (v2.0) needed many improvements, not only because of missing functionalities but also because of the ambiguities introduced. In particular, the WLCG communities of researchers needed explicit and specific functionalities (as mentioned in Chapter 3) in order to be able to implement their physics analysis and production software environment. Therefore, v2.1 of the SRM protocol was proposed and WLCG developers took active part in the discussions in order to satisfy the user requests (see Section 3.5.2).

At the CHEP 2006 conference in Mumbai, conveners agreed that the available v1.1 and v2.1 SRM implementations did not provide the needed functionalities. Therefore, researchers met at the Fermi National Accelerator Laboratory (FNAL) in Chicago in May 2006 in order to define the design of SRM v2.2. In September 2006 the DoE and the National Science Foundation (NSF) in the USA decided to fund substantially the Open Science Grid (OSG) project. This contributed to create momentum around the SRM working group and the collaboration became strongly and actively international. As of today, v2.2 of the protocol is being finalized. In order to guarantee a timely implementation of the protocol in all available storage services in WLCG, a reduced version of the v2.2 protocol has been proposed. Among other restrictions, the WLCG version 2.2 of SRM will initially allow for static or "restricted" dynamic reservation only. Various testing suites have been developed in order to verify the functionality of the proposed interface, the conformity to the standard and the interoperability of the implementations. Weekly meetings are taking place between EU and US Grid SRM developers in order to discuss and finalize v3.0 of the SRM interface, with the possibility of exposing some of the new needed features in v2.3 that could be already available by the end of 2007.

6.2 The Storage Resource Manager Interface

*A **Storage Resource Manager*** (SRM) *is a middleware component whose function is to provide dynamic space allocation and file management on shared storage components on the Grid* [81].

In what follows we define a model that describes the SRM interface version 2.2. We introduce the main concepts assumed by the interface and the functionalities offered.

An SRM is a standardized interface (a detailed specification for version 2.2 can be found in [82]) offered by a Storage Element that satisfies most of the requirements explained in Section 6.1.2. It allows authorized clients to initiate requests for allocating and managing a given amount of space with a given quality. The space can be used to create new files or access existing files with a set of negotiated protocols stored in the underlying storage system.

The granularity of the data that SRM can refer to has been established to be the file, since more specific granularities such as "partial files" or "objects" implies a knowledge on the structure of the files being managed by the SRM. Such knowledge is application dependent and cannot be generalized.

To illustrate the functionality of an SRM, we introduce the following example: a user job has to generate a given output file. In order to be successful, the job has to make sure that the amount of space needed for storing the output produced is available and can be guaranteed to the owner of the job. Therefore, after authentication and authorization are performed (point 1 of the requirements) the job can issue an SRM request to allocate or reserve a certain amount of space (requirement 5) with a certain quality (magnetic device/online disks) (requirements 4 and 6) at a Storage Element accessible from the Computing Element where the job is executed. Additionally, the application running on the Computing Element is enabled to only use a certain set of file access or transfer protocols (requirement 10). Therefore, via the SRM interface, the application can negotiate the provision of an area of space where the requested protocol can be used. As a result, the SRM provides a protocol dependent file handle (see next section) that the application can use to write the output file. Other user jobs can check for the availability of the produced output file in order to start further processing. Therefore, the SRM interface provides for a pinning mechanism, filesystem like operations (requirement 5) and mechanisms for transparently locating data (requirement 7). Temporary copies of the output file can be created for optimization reasons as volatile data (requirement 6).

The space reserved by the job in the example above can have a lifetime that can be finite or infinite. After the lifetime is expired, the space can be recollected by the system and reused to satisfy other requests. Files also have a lifetime and, if not permanent, they can be removed by the system. Additionally, files can have many copies in the system. Such copies have a lifetime ("pin" lifetime). When the lifetime expires, the copies might be automatically removed by the system. Copies can have a protocol specific handle with a given lifetime. Clients can use such a handle to access the file's copies in read mode or to overwrite an existing copy.

In order to logically organize and manage files, the SRM offers a ***namespace*** similar to the one of a UNIX filesystem. Under a root directory, users can create subdirectories and files. As a result, applications can refer to these files via UNIX-like paths.

The SRM interface is implemented using a Web Service described by the WSDL v1.1 specification published in [83]. It uses SOAP over HTTP. In order to allow for authenticated and authorized access, GSI is used.

6.2.1 The SRM v2.2 methods

The v2.2 SRM interface functions or methods are described in detail in [82] and can be categorized in five families: space management functions, permission functions, directory functions, data transfer functions, and discovery functions.

Some of these methods might be asynchronous. In such cases, they return a ***request token*** that the user has to use with subsequent query methods in order to check the status of the request. HEP applications can normally access many files at once. In fact, different kind of physics information is stored in different files. In order to reconstruct a physics event, a user needs to be able to open multiple files at once. Therefore, the SRM interface allows users to request operations on a set of files (data-set). In this case, the request token allows users to query the status of or terminate an on-going operation that involves a set of files. For methods that accept as input a set of files, the call returns a status code at request level and a status at file level, for each of the files included in the request.

Let us describe the functionality offered by the different families of SRM functions.

6.2.1.1 Space management functions

Space management functions allow the client to reserve, release, and manage spaces, their types and lifetime. Once the space is assigned, it can be referred to with a space token. The main space management functions are:

- ***srmReserveSpace***: it allows the requester to allocate space with certain properties. Once the space has been allocated, a space token (an identifier of the space) is returned and the requester becomes the owner of the space. Only the owner can release the space. In order to be able to guarantee the space reserved, it is assumed that the underlying storage system can enforce a quota limit at least at request level: once a user has reserved a space, he/she could store files in this space for an amount that is much greater than reserved. In order to prevent this abuse of the system, it is assumed that the system can enforce some kind of quota checking so that a user can never exceed 10% of its assigned space.
  This call might be asynchronous since some systems may need to perform various operations in order to guarantee the space required by concurrent requests. In particular, if no space with the requested properties is available, the system might choose not to fail and to retry later when some files have been garbage collected. In this case, the request stays queued.
  The space reserved might be contiguous or not. The user can specify the number and sizes of the files that the space will hold in order to allow the storage server to perform some optimization and reserve the space appropriately.
- ***srmUpdateSpace***: spaces are characterized by a size and a lifetime, which is the time for which the space is allocated and granted to the requester. This method allows the owner of a space to increase or decrease its size and lifetime. This call might be asynchronous and the same considerations made for *srmReserveSpace* apply.
- ***srmReleaseSpace***: it releases an occupied space. The call is synchronous, however it might not complete immediately. In fact, if the space contains copies of a file, the system must check if those copies can be deleted in order to reuse the space.
- ***srmChangeSpaceForFiles***: it is used to change the space where the files stay. A target space token must be specified. The new space will have different properties with respect to the original one. All files specified in the request will have a new space token. The file path does not change when the space for a file changes. In general copies of files can be in multiple spaces at one time.

This is achieved via SRM methods such as *srmPrepareToGet* or *srmBringOnline* (described later). This method might be asynchronous. Therefore, a request token is returned.

- ***srmExtendFileLifeTimeInSpace***: it is used to extend the lifetime of files that have a copy in the space identified by the space token passed as input argument. The new lifetime cannot exceed the lifetime of the space in question.
- ***srmPurgeFromSpace***: this function is used to remove copies of files in a space. If the specified input non-expired or permanent file has its last copy in the space, then an error is returned.

6.2.1.2 Directory functions

Directory functions are similar to their equivalent UNIX functions. The SRM interface offers a namespace, which is similar to the one of a filesystem. In particular, files have a path and a filename. Details will be given later on in Section 6.3. The SRM directory functions allow users to put files with different storage requirements in a single directory. Changing the space characteristics of a file does not change its position in the directory but only its assignment to the corresponding space. The directory functions are:

- ***srmMkdir***: it creates a directory in the SRM namespace. Some SRM might have automatic directory creation enabled: when a user specifies a new file and the file path has not yet been created, the storage system creates it automatically. This is for instance the case for the current implementation of the dCache SRM. However, users are encouraged to explicitly use this function in order to avoid "polluting" the namespace by mistake.
- ***srmRmdir***: it removes empty directories in a local SRM namespace.
- ***srmRm***: it removes files from the SRM. The storage system will remove automatically the file entry in the namespace. The system will automatically mark all existing copies of the files as invalid and therefore make them candidate for garbage collection. This call cannot be used to remove directories.
- ***srmLs***: it returns a list of files with basic information. This call can be asynchronous. Even though the call allows for a number of entries to be returned and an offset to be specified in case of directory listings, none of the SRM implementations at the moment support such a feature since the server is stateless. Once the stateful capabilities of WSRF will be adopted, it would be easier to allow for such a feature.
- ***srmMv***: this call allows users to change the path of a file. The file will be "moved" within the SRM namespace. The function applies to both directory and files.

6.2.1.3 Permission functions

Permission functions allow a user to assign read and write privileges on a specific file to other users, or reassign files to other users. Such functions allow client applications to specify ACLs as well, wherever supported by the underlying storage service.

- ***srmSetPermission***: it is used to set permissions on both a file or a directory. Permissions can be assigned to a set of users or a set of groups. It is assumed that groups are predefined. Users can either be identified via their X.509 DNs or via their mapped UNIX IDs.
- ***srmCheckPermission***: it is used to check the client permissions on the files. It returns the permissions for the specific client.

- ***srmGetPermission***: it is used to get the permissions on the file. It returns the permissions of owner, users, groups and others.

6.2.1.4 Data transfer functions

Data transfer functions have the purpose of getting files into SRM spaces either from the client's space or from other remote storage systems on the Grid, and to retrieve them. Data transfer functions support requests for multiple files within a single transaction. These functions constitute the core of the SRM specification. The main methods are the following ones:

- ***srmPrepareToGet***: it returns a protocol specific handle to an online copy of the requested file. A pinning expiration time is assigned to this handle. When a space token referring to an online space is passed as input argument, the handle returned by this function refers to a copy of the file created in the space associated with the space token. In WLCG it has been decided that any single file can only be in one space at a given time [120]. This restriction is enforced by ignoring the space token passed as input argument to this method.
- ***srmBringOnline***: it is used to make files ready for future use. If the requested files are stored on tape, the system might spool them on disk in order to create file handles. The spooling operation is time consuming and therefore the client can schedule it in advance, before the file handles are really needed. Therefore, this call allows clients to specify a deferred time when the files have to be ready. A target space can also be specified where the files can be made available. However, in WLCG such a space token is ignored [120]. This operation is asynchronous.
- ***srmPrepareToPut***: it creates a protocol specific handle with a pin lifetime that clients can use in order to create new files in a storage space or overwrite existing ones. When a specified target space token is provided, the files will be located finally in the targeted space associated with the target space token. Otherwise, a new file is created in a default space with the requested properties specified as input arguments to this method. It is an asynchronous operation.
- ***srmPutDone***: it tells the storage system that the write operations are done. The corresponding files are associated to handles allocated via a previous *srmPrepareToPut* request. The handles can be released and their pinning lifetime set to zero. The lifetime of the respective files starts when this call is invoked.
- ***srmCopy***: it allows a client to create a file by copying it in the SRM space. The target file can be local with respect to the SRM storage service or can be in a remote SRM. In case of a remote copy, the operation can be performed in *PULL mode*, i.e. the target SRM is the one that serves the client *srmCopy* request, or *PUSH mode*, i.e. the source SRM is the one that serves the client's request. These modes are necessary in order to be able to perform operations even when the target is behind a firewall. When a space token is specified, the files will be located finally in the targeted space associated with the space token. The call is asynchronous.
- ***srmReleaseFiles***: it sets to zero the pin lifetime of the copies of a file generated by an *srmPrepareToGet* or *srmBringOnline*. The system can then remove the copies if the space is needed for other files.
- ***srmAbortRequest/srmAbortFiles***: Many SRM methods are asynchronous and therefore the respective requests return a request token that can be used to check the status of the request or to terminate prematurely. The effect of these methods depends on the type of request. For a data transfer request, the SRM will attempt a complete cleanup of running transfers and files in intermediate state.
- ***srmExtendFileLifeTime***: this method is used to extend the lifetime of non-permanent files or the pin-lifetime of TURLs or copies.

6.2.1.5 Discovery functions

Finally, ***discovery functions*** allow applications to query the characteristics of the storage system behind the SRM interface and the SRM implementation itself. There are only two discovery functions:

- ***srmPing***: it is used to check the status of the SRM.
- ***srmGetTransferProtocols***: this function is used to discover which file access and transfer protocols are supported by the Storage Element.

6.3 The Data and Semantic Model

The current specifications for the SRM version 2.2 do not define a complete semantic model. Furthermore, the entities on which the SRM interface operates are not clearly defined. In the past, this has created quite some difficulties in mapping the concepts assumed by the interface with the physical and logical entities and functions offered by the underlying storage systems. Here, we define the data model and the protocol behind the SRM interface as the ordered set of interactions between client and server. In order to define a model for SRM, let us introduce the concepts of ***spaces***, ***files***, ***copies***, and ***handles***. We first introduce the basic definitions and a mathematical formalization of these concepts and then we elaborate on them in more details.

6.3.1 Basic definitions

In order to describe the SRM in terms of elementary mathematical concepts, we first introduce some basic sets whose members are unstructured values, such as atomic symbols (meant to represent names of objects or discrete values for their attributes) or numbers. Then we define the constructed sets of storage elements, spaces, copies, handles, and files as Cartesian products of some basic sets. Hence, each element of one of these constructed sets is a tuple with named components. We call the components attributes, and we use the dot notation to refer to the value of an object's component. For example, if a set S is defined as B_1 X B_2, its elements are tuples of the form ⟨attr1, attr2⟩, with attr1∈B_1 and attr2∈B_2. If an object o belongs to S, the expression o.attr1 denotes the value of its first component.

All sets defined in the following are mutually disjoined.

Only the most relevant attributes will be considered in this formalization. The corresponding basic sets will be introduced incrementally, i.e., when defining a new constructed set we will mention only the basic sets that have not been previously introduced.

6.3.1.1 Storage Element

We define the following basic sets:

Storage Element identifiers	*SEid*		a finite (countable) set of symbols
Sizes	*Sz*	=	**IN**
Retention quality	*Rq*	=	{**REPLICA**, **OUTPUT**, **CUSTODIAL**}
Access latency	*Al*	=	{**ONLINE**, **NEARLINE**}
Protocols	*P*	=	{**rfio**, **dcap**, **gsiftp**, **file**}
Access Pattern	*Ap*	=	{**TRANSFER**, **PROCESSING**}

Connection Type	Ct	=	{**WAN**, **LAN**}

For sets Rq and Al we have respectively:

REPLICA < **OUTPUT** < **CUSTODIAL**

ONLINE < **NEARLINE**

We define the set of *policies* as:

(6.3.1) $$Pol = Rq \times Al$$

And a *retention policy* (referred also as *retention policy info*) as a tuple of the form

(6.3.2) ***rp = <retqual, latency>***

We define the set of *storage properties* as:

(6.3.3) $$Prop = Pol \times P \times Ap \times Ct$$

And the *storage element properties* as a tuple of the form:

(6.3.4) ***seprop = <policy, protocol, access, connection>***

The set of supported properties is the powerset of the storage properties:

(6.3.5) $$Sprop = \mathcal{P}Prop$$

We finally define the set of *storage elements* as:

(6.3.6) $$SE = SEid \times Sprop \times Sz$$

A *storage element* is a tuple of the form:

(6.3.7) ***se = <id, sprops, size>***

6.3.1.2 Space

We define the following basic sets:

Space Tokens	T		a finite (countable) set of symbols
Lifetimes	L	=	**IN** $\cup$ **{**T**}**
Owners	O		a finite (countable) set of symbols
Space Requests	R_s		a finite (countable) set of symbols

For set L, we have:

$$t<T,\ \forall t \in L$$

where T (*top*) denotes an unlimited value.

Other properties:

Space Token Description	*TD*	a finite (countable) set of symbols ∪ {**NULL**}

where **NULL** represents an unspecified value.

We finally define the set of spaces as:

(6.3.8) $S = T \times L \times Prop \times Sz \times O \times RS \times TD$

A *space* is a tuple of the form:

(6.3.9) ***s = <token, lifetime, props, size, owner, request, tokendescr>***

6.3.1.3 Copy

We define the following basic sets:

Physical File Names	*Pfn*	a finite (countable) set of symbols
Copy requests	R_c	a finite (countable) set of symbols ∪ {**NULL**}

where **NULL** represents an unspecified value.

We define the set of copies as:

(6.3.10) $C = Pfn \times L \times R_c$

A *copy* is a tuple of the form:

(6.3.11) ***c = <physname, pintime, request>***

6.3.1.4 Handle

We define the following basic sets:

Physical File Names	T_r	a finite (countable) set of symbols
Handle requests	R_h	a finite (countable) set of symbols

We define the set of handles as:

(6.3.12) $H = T_r \times L \times R_h$

A *handle* is a tuple of the form:

(6.3.13) ***h = <turl, pintime, request>***

6.3.1.5 File

Finally, let us introduce the definition for a file. We define the following basic sets:

We define the following basic sets:

SURLs	Sr		a finite (countable) set of symbols
File Types	Ft	=	**{FILE, DIR}**
File Status	Fs		a finite (countable) set of symbols
Creation Time	T_c	=	**IN**
Modification Time	T_m	=	**IN**
File Storage Types	St	=	{**VOLATILE**, **DURABLE**, **PERMANENT**}
File Locality	Fl	=	{**ONLINE**, **ONLINE_NEARLINE**, **NEARLINE**, **UNAVAILABLE**, **LOST**}
Permissions	Pm		a finite (countable) set of symbols

For sets St and Fl we have respectively:

VOLATILE < **DURABLE** < **PERMANENT**

ONLINE < **ONLINE_NEARLINE** < **NEARLINE** < **UNAVAILABLE** < **LOST**

We define the set of files as:

(6.3.14) $F = Sr \times Ft \times Fs \times T_c \times T_m \times St \times L \times Pol \times Fl \times Pm$

A $file$ is a tuple of the form:

(6.3.15) ***f = <surl, ftype, status, size, ctime, mtime, stype, lifetime, policy, locality, perm>***

6.3.2 Functions and relationships

We define the following functions and relationships:

`se` A space is hosted on a *se*.

$$se: S \rightarrow SE$$

`spaces` Function *spaces* returns the set of spaces hosted by a given se.

$$spaces: SE \rightarrow S$$

`stime` Function *stime* is the start time of a space, copy, handle or file, i.e. the time when its (pin) lifetime starts to be counted down.

$$stime: S \cup C \cup H \cup F \rightarrow \mathbf{IN}$$

`lleft` Function *lleft* is the remaining (pin) lifetime of a space, copy, handle or file at a given time.

$$\texttt{lleft: } (S \cup C \cup H \cup F) \times \mathbf{IN} \rightarrow \mathbf{IN}$$

`file` Function *file* gives the file owning a copy.

$$\texttt{file: } C \rightarrow F$$

`fcopies` Function *fcopies* gives the set of copies of a file.

$$\texttt{fcopies: } F \rightarrow \mathcal{P}C$$

`space` A copy is hosted on a *space*.

$$\texttt{space: } C \rightarrow S$$

`scopies` Function *scopies* gives the set of copies hosted by a space.

$$\texttt{scopies: } S \rightarrow \mathcal{P}C$$

where:

$$c \in scopies(s) \Leftrightarrow s = space(c)$$

`refcopy` Function *refcopy* gives the copy that is referred to by a handle.

$$\texttt{refcopy: } H \rightarrow C$$

`fhandles` Function *fhandles* gives the set of copies of a file.

$$\texttt{fhandles: } F \rightarrow \mathcal{P}H$$

where:

$$h \in fhandles(f) \Leftrightarrow \exists c(c \in fcopies(f) \wedge c = refcopy(h))$$

`shandles` Function *shandles* gives the set of handles that refer to a copy held by a space.

$$\texttt{shandles: } S \rightarrow \mathcal{P}H$$

where:

$$h \in shandles(s) \Leftrightarrow \exists c(s \in space(c) \wedge c = refcopy(h))$$

`master` A file has one *master* copy.

$$\texttt{master: } F \rightarrow C$$

`resfiles` A file is *resident* on a space if the space holds the file's master copy. Function *resfiles* gives the set of files resident on a space.

$$\text{resfiles}: S \rightarrow \mathcal{P}F$$

where:

$$f \in \text{resfiles}(s) \Leftrightarrow \exists c(s \in \text{space}(c) \wedge c=\text{master}(f))$$

mspace The space holding a file's master copy.

$$\text{mspace}: F \rightarrow S$$

6.3.3 Constrains

In the following, *se* denotes the storage element. Then the following constrains hold:

Space

size

(6.3.16) $$\sum_{s \in \text{spaces}(se)} s.\text{size} \leq se.\text{size}$$

properties

(6.3.17) $$\forall s \in \text{spaces}(se),\ s.\text{props} \in se.\text{props}$$

Copy

integrity

(6.3.18) $$\forall c \in C\ \exists se \in \text{SE},\ \exists_1 s \in \text{spaces}(se):\ c \in \text{scopies}(s)$$

(6.3.19) Given se, $\forall c \in C$ such that $\exists_1 s \in \text{spaces}(se)$ with $c \in \text{scopies}(s)$
$$\exists_1 f \in F:\ \text{mspace}(f) \in \text{spaces}(se) \wedge c \in \text{fcopies}(f)$$

pintime

(6.3.20) Given se, $\forall c \in C$ such that $\exists_1 s \in \text{spaces}(se)$ with $c \in \text{scopies}(s)$
$$0 < c.\text{pintime} \leq \min(\text{space}(c).\text{lifetime},\ \text{file}(c).\text{lifetime})$$

Handle

integrity

(6.3.21) $$\forall h \in H\ \exists se \in \text{SE},\ \exists_1 s \in \text{spaces}(se):\ h \in \text{shandles}(s)$$

(6.3.22) Given se, $\forall h \in H$ such that $\exists_1 s \in \text{spaces}(se)$ with $h \in \text{shandles}(s)$
$$\exists_1 f \in F:\ \text{mspace}(f) \in \text{spaces}(se) \wedge h \in \text{fhandles}(f)$$

pintime

(6.3.23) Given se, $\forall h \in H$ such that $\exists_1 s \in \text{spaces}(se)$ with $h \in \text{shandles}(s)$
$$0 < h.\text{pintime} \leq \min(\text{space}(\text{refcopy}(h)).\text{lifetime}, \text{file}(\text{refcopy}(h)).\text{lifetime})$$

File

integrity

(6.3.24) $$\forall f \in F \; \exists se \in \text{SE}, \; \exists_1 s \in \text{spaces}(se): \; f \in \text{resfiles}(s)$$

policy

(6.3.25) Given se, $\forall f \in F$ such that $\exists_1 s \in \text{spaces}(se)$ with $f \in \text{resfiles}(s)$
$$f.\text{policy} \in se.\text{sprops.policy} \wedge$$
$$f.\text{policy.retqual} = s.\text{props.policy.retquality} \wedge$$
$$f.\text{policy.latency} \geq s.\text{props.policy.latency}$$

lifetime

(6.3.26) Given se, $\forall f \in F$ such that $\exists_1 s \in \text{spaces}(se)$ with $f \in \text{resfiles}(s)$
$$0 < f.\text{lifetime} \leq s.\text{lifetime}$$
$$\exists t > \text{stime}(f), \; \text{lleft}(f,t) > 0$$

6.4 Detailed discussion on the model

In what follows we elaborate on the formalized model in order to better describe it. We also provide UML class, state, and activity diagrams of the entities previously introduced and study some of their interactions.

6.4.1 The Space

We have already provided an overview of the Storage Element service and the functionality provided (see Chapter 2). Let us describe the concepts associated with a ***space***. As shown in section 6.3, a Storage Element (SE) consists of a set of spaces. A space is characterized by a set of attributes that are exposed to users. The essential public attributes are defined in (6.3.8) and (6.3.9).

A ***Space*** s *is an area of storage that can be statically (via an explicit requests to the administrators of a site) or dynamically (via SRM calls) allocated by clients*. Once reserved, a space can hold files. Default spaces with specific characteristics and attributes could be provided by the system.

A space has a ***lifetime***, a time period for which the space exists. It can be limited or unlimited. If the lifetime of a space is unlimited, then we say that the space is ***permanent*** (cf. definition of Lifetimes in Section 6.3.1.2). After the lifetime of a space is expired, the space can be reclaimed by the system and specific actions are defined for the data contained in such a space. Spaces are characterized by a ***retention quality*** (***retqual***) that refers to the probability that the space loses a file (see (6.3.2)). The possible values are REPLICA, OUTPUT, CUSTODIAL. REPLICA has the highest probability of lost, while custodial

has the lowest. OUTPUT quality has an intermediate probability. In REPLICA space it is stored data that can be lost. In the latter case, data might exist in multiple copies in the system, in spaces with different retention quality capabilities. CUSTODIAL space holds the master exemplar of some data that must be preserved. In OUTPUT space data that are lost can be recovered with some lengthy or expensive processes. The SE can support all or only some of the possible retention qualities.

The ***access latency*** (***latency***) describes how latency can be improved. ONLINE provides users with the lowest latency. Data in a NEARLINE space can be accessed more rapidly moving them in an ONLINE space (or cache). Also in this case the SE can support all or only one of the possible allowed latencies.
A space has an ***owner***, who is the entity that has created the space and that can release the space. The DN identifies the owner. The space is also characterized by a ***size***. In particular, the ***total size*** of a space refers to the size that will be given by the storage system on a best effort base: if space is available at the time the client requests to store data in the space reserved, then the space will be given. The guaranteed size is the size that the system has allocated to that space. There are in fact 2 models for space allocation: one model is to negotiate the amount of space that the client can have. We refer to this model as the ***on-demand*** model. The other possibility is to have the SRM allocate a certain amount of space to the client according to a quota determined by its policy. Then, the SRM brings in files up to the quota level. Only when the client releases one of the files, SRM will schedule another file to be brought in provided that the total space used is below the quota. We call this the ***streaming*** model.

Clients may refer to a combination of spaces properties via a ***Space Token Description***. There could be many spaces corresponding to a given Space Token Description. Therefore, Space Token Descriptions are not unique in an SRM storage system. A ***Space Token*** identifies each of those space instances and it is unique and immutable. The internal properties of a space listed above describe the capability of a space to be accessed for specific access patterns and in specific network domains with given supported protocols.

The ***Access Pattern*** internal attribute specifies allows users to specify the nature of the operation that will be performed on the space, and therefore allows a system to choose physical buffers to assign to that space with characteristics that allow for optimization of the operations. The space can therefore be used to perform transfer operations (TRANSFER_MODE access) or for reading/[over]writing files (PROCESSING_MODE access). Spaces that support TRANSFER_MODE access can perform some optimization in order to privilege transfer operations: for instance they can provide disks configured to use large TCP window buffer sizes. PROCESSING_MODE instead might privilege frequent use of "seek" operations.

The ***Connection Type*** specifies if a space is visible to the LAN only or to the WAN. SRM may optimize the access parameters to achieve maximum throughput for the connection type. The values that this parameter can assume are sometimes misleading. There are situations where the storage service is located at a different location and even in a different network domain with respect to the computing facilities. This is the case of the NIKHEF facility in The Netherlands. NIKHEF provides mostly computing resources while the storage resources are served by the SARA facility, located in a different organization with a different Internet domain. In this case, only the computing facilities at NIKHEF can access the storage services available at SARA from a Grid environment. The SARA storage resources are therefore not accessible via WAN in the generic sense, but are available to the nodes at NIKHEF. In this case, the Connection Type parameter has to be set to LAN for SARA, even if the resources are available via WAN to the NIKHEF computing facility.

Protocols is a list of file access and transfer protocols supported by this space. When making a request to an SRM, the client needs to use a protocol for accessing/transferring the file that the storage system or

even the specific reserved space can support. In general, systems may be able to support multiple protocols (the corresponding host runs the daemon implementing the specific protocol) and clients may be able to use multiple protocols depending on the system they are running on. The client can negotiate with the SRM server a list of supported protocols and choose one. The most used file access and transfer protocols are described in the next chapter. Internal policies might establish that the use of some of these protocols is restricted to certain VOs or VO FQANs (as defined in Chapter 2).

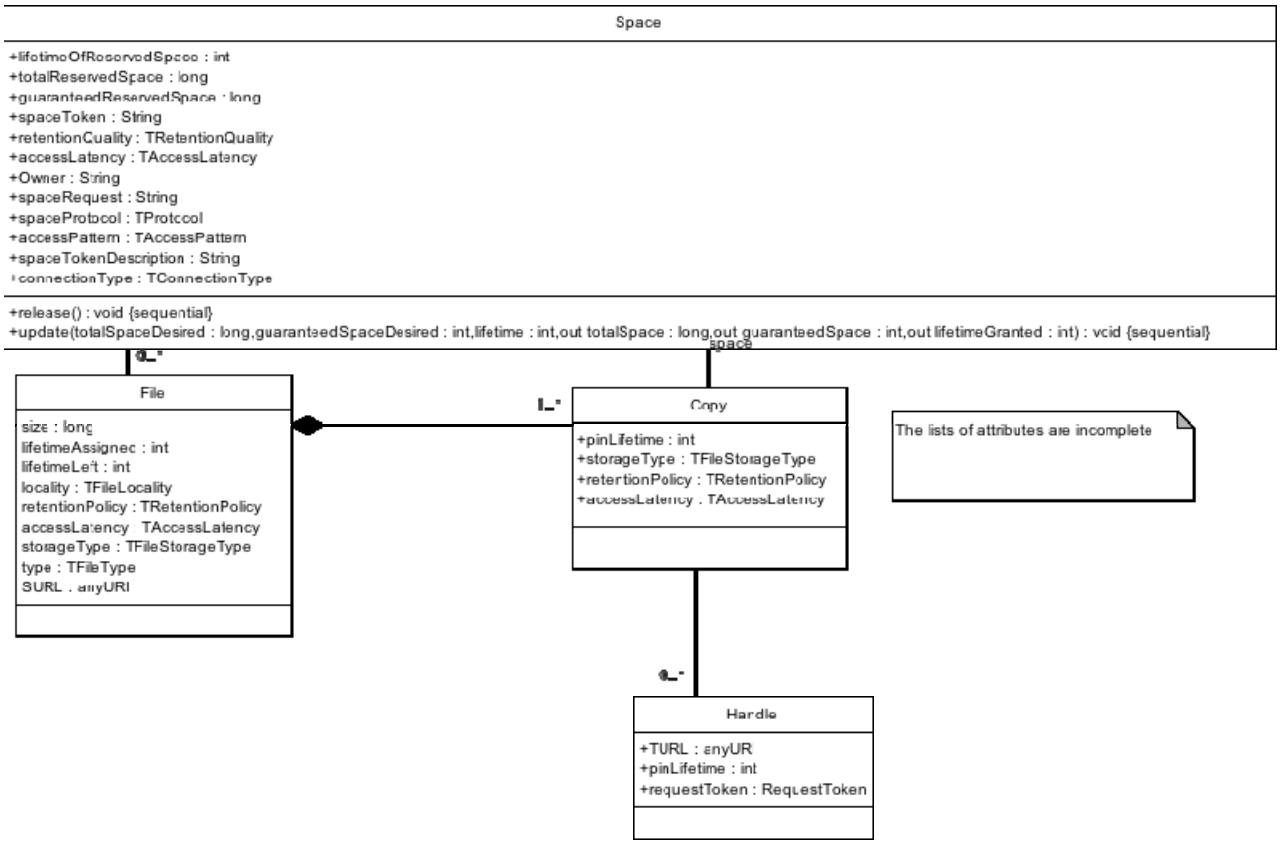


Figure. 6.1 The SRM v2.2 Space, File, Copy, and Handle UML class diagram

Other features specific to a particular implementation can also be specified when reserving spaces on a Storage Element. As an example we can mention the possibility of using a specific tape set to store data in an SRM space permanently on tape (in dCache this possibility is given for instance by specifying the Storage Group [64]).

Figure 6.1 shows a UML description of the Space in SRM 2.2. It shows the relation of the space class with the other SRM classes: files, copies and handles as described in what follows.

Clients can perform various operations on spaces. In particular, spaces can be created, updated enlarging their size and extending their lifetime. Spaces can also be released or removed by the owner of the space or by users with specific privileges. Figure 6.2 shows the UML state diagram for the space in SRM 2.2.

A user reserves a space with a given size and for a period of time (that can be infinite) invoking *srmReserveSpace*. Such a space is commonly assumed to be online (disk) since the space on more reliable media can be partitioned and reserved statically in advance. However, the SRM model makes no explicit

assumptions about this matter. The reserved space might not be contiguous. Therefore, the user can specify the number and size of the files that should occupy that space so that the system can optimize the allocation. The *srmReserveSpace* call can also be implemented as a static function in the sense that only authorized users can use it, and it triggers a request to an operator that manually performs the operations needed to guarantee the given space. This is the case of the CASTOR system at CERN.

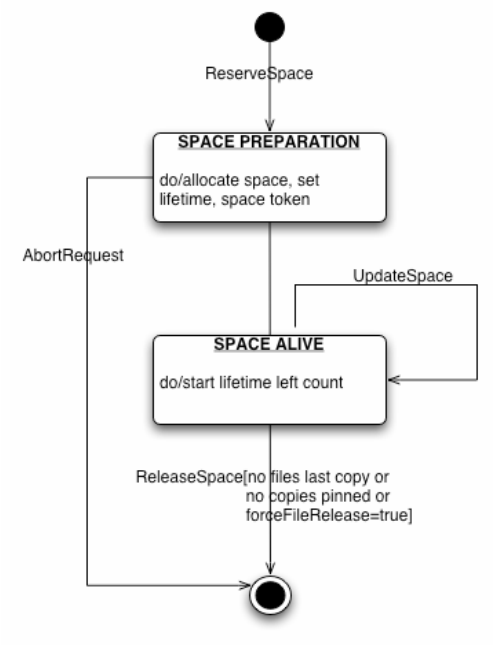


Figure. 6.2 The SRM v2.2 Space UML state diagram

As mentioned above, spaces can have certain characteristics that the user can specify at allocation time in order to obtain the necessary quality of service for his/her files. Once allocated, the space can be populated with copies of files. If needed, spaces can be enlarged or their lifetime can be extended. This is achieved via the SRM method *srmUpdateSpace*. The system however can always negotiate with the client the space guaranteed and its lifetime. Only the owner of a space can release it. However, special user identification parameters can be passed as an input argument to the SRM calls so that local policies can be applied. At the moment, no implementation allows users of a given group to perform space allocation in a space already reserved to a given VO. In fact, this feature is not allowed by the SRM specifications since the space token cannot be passed as input argument to the *srmReserveSpace* method in order to specify the space where the second sub-allocation should take place. However, this possibility is strongly needed by the LHC VOs and will be made possible with version 3 of the SRM specifications.

SRM methods to inquire the characteristics and status of spaces by specifying the correspondent space tokens are available. However, users should keep track of the space tokens that the system has assigned to the spaces. It is not possible, in fact, to use only the space token descriptions. The SRM storage systems do not manage the spaces in the sense that it is up to the client applications to decide which space to use and which one is more adequate for certain operations.

Reserved spaces can be released and the occupied size returned to the storage system. This is achieved through the SRM method *srmReleaseSpace*. However, the copies of the files contained in these spaces can either be deleted or not, depending on their type. This concept will be clarified later on in this chapter.

The SRM method *srmAbortRequest* can be used to interrupt all asynchronous actions on spaces. Everything described is represented in the class state diagram for a space in Figure 6.2.

6.4.2 Files, Copies and Handles

A ***file*** *is a set of data stored within the storage system with an associated path and properties*. It has been formally defined in (6.3.14), (6.3.15), (6.3.24), (6.3.25), (6.3.26).

It is associated bi-univocally to an SURL, as defined in Chapter 2. An ***SURL*** *or Site URL is a URI* [RFC 2396 and 2732] *that provides information about the physical location of the file since the hostname of the SRM server is coded in the SURL*. It consists of

srm://host[:port]/[soap_end_point_path?**SFN=**]site_file_name

where everything contained in square brackets is optional and the strings in bold are required.

The ***SFN*** or Site File Name *is a full file name* (directory name/file name*) assigned by a site to a file.*

For optimization reasons, a file might have many copies in a Storage Element. These copies can be contained in different types of spaces and even in multiple storage resources. Furthermore, it is convenient to be able to move files from one storage device to another one without changing the SFN that is directly exposed to applications. Thus, the SFN does not have to correspond to the physical storage location of the file.

The ***StFN*** or Storage File Name is a string that contains the file path/name of the intended storage location when a file is put (or copied) into an SRM controlled space. Thus, a StFN can be thought of a special case of an SURL, where the protocol is assumed to be "srm" and the machine:port is assumed to be local to the SRM. The StFN is also known as ***PFN*** or Physical File name.

Finally, the ***TURL*** or Transport URL is a URI that gives the necessary information to retrieve the associated physical copy, including hostname, protocol and port, path, so that the application can open or copy it.

In order to model the different entities that we have just described, let us consider the following use case.

An application requests to create a file in a given reserved space in the Storage Element. The system assigns an entry in the SE namespace (SURL) to it and provides the application with a file handle to write data in the space reserved for the requested file. Depending on the class of storage (i.e. retention policy) requested for this data file after the file has been written and closed the system might migrate to nearline space (for instance, to a tape system). When the user/application accesses again the file, it demands for the data to be available online on some disk server for further processing. However, the data might be processed later on. Therefore, a copy of the file is made available online for a given period of time (copy pin-lifetime) on some disk server for later processing. The user does not yet have any handle to access or use this copy.

A ***copy*** `c` *is an instance of a file* `f` *in a given space* `s` *with an associated* `Physical File Name` *and a pin-lifetime*. A copy has been formally defined in (6.3.10), (6.3.11), (6.3.18), (6.3.19), (6.3.20).

Copies have a pin-lifetime that never exceeds the lifetime of the space that contains them or the lifetime of the file they refer to (see constrain (6.3.20)).

Let us consider again our use case. In order to make such a copy available, the system might schedule the operation of making an online copy of a file. This might imply reserving some tape drive to read the associated data from tape to disk. This operation may require some time. Therefore, the client can ask for the file to be online in advance. An StFN (or Physical File Name, PFN) identifies the copy. Once the application is ready to access the file, a protocol is negotiated with the SRM for file access. This protocol can be GridFTP if the file needs to be transferred somewhere else, or some remote file access protocol (described in the next chapter). It could happen that for optimization reasons another physical copy of the file had already been created in some disk buffer on the system with the characteristics requested by the application: the server serving that buffer is for instance accessible on the WAN for transfer operations, the TCP/IP window size has been set to optimize the transfer of large buffers, or the server runs an instance of the daemon for the remote file access protocol requested by the application. Such a copy is therefore accessed via a protocol dependent handle that the application can use to access the file for the lifetime of the handle (TURL).

A ***handle*** *is a user-guaranteed transport specific URL (TURL) with associated pin-lifetime and an online copy of a file in a space. A handle is always associated to an SRM request. This expresses the characteristics of the space, a user-created one or the default, where the copy of the file associated to the corresponding TURL lays.* A handle has been formally defined in (6.3.12), (6.3.13), (6.3.21), (6.3.22), (6.3.23).

Handles have a pin-lifetime that never exceeds the lifetime of the space that contains them or the lifetime of the file they refer to (see (6.3.23). It specifies the period of time for which the file handle (TURL) must exist, as per agreement with the application that has requested it. Copies of a data file in a space do not have to be the same as the ones associated with Handles. This is left to the underlying storage system to manage. Figure 6.3 depicts this use case.

In Figure 6.3 the file "9" has been created in the Storage Element with infinite lifetime and a near-line copy is on tape. A user asks the system to make a copy of "9" available online on some disk servers for later use. The copy will be used not before 2 hours and for no longer than 5 hours from the time the request has been made. The system identifies the tape where the near-line copy resides and schedules the allocation of a tape drive where to read the relative data from. Two hours later, the online copy is available for the user as requested.

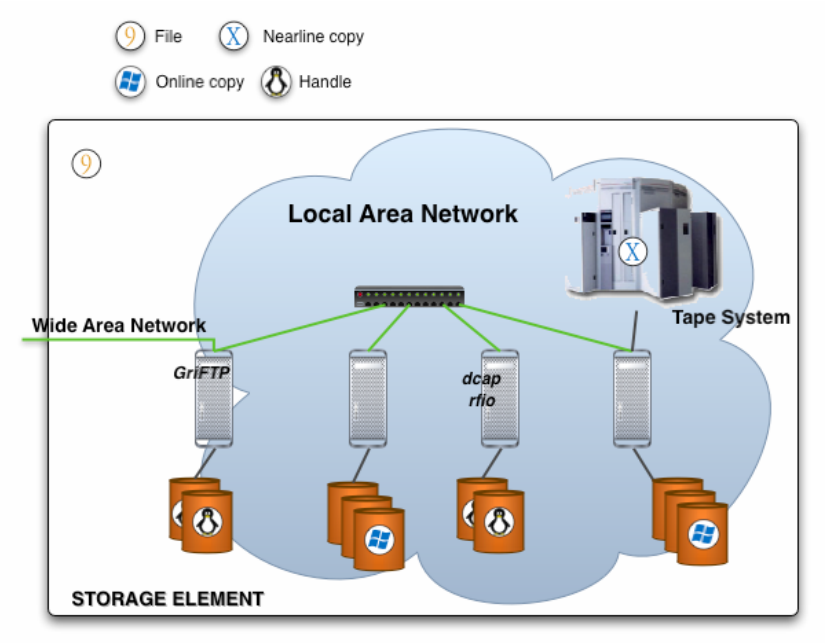


Figure. 6.3 Copies and handles of a file in a Storage Element

Now the application requests for 4 handles/TURLs to be available, one to access the data with the rfio data access protocol, one for file access with the dcap protocol and 2 for data transfer via the GridFTP protocol. The first 2 TURLs should be available for only 1 hour, while the other 2 for 5 hours. For internal optimization reasons, the system creates another internal copy of the file on a different server with pin-lifetime of 3 hours. Then it starts the internal data movers to create copies on the data servers where the GridFTP, rfio and dcap daemons run, to make available the relative Handles. While the data file access and transfer take place the copies of the file "9" with no TURL associated are removed to make space for other requests.

Let us now describe in more details some of the file attributes and we refer to the v2.2 specification document for the others left out.

A files has an associated ***path***, which is the absolute path assigned by the site to that SURL. Together with the filename, it defines the SFN of the file.

The first instance of a file is defined to be the ***primary copy*** or ***master***. A file has an associated ***Lifetime*** that cannot exceed the lifetime of the space where the primary copy was created (master space). This lifetime can be shortened or extended but they can never be greater than the lifetime of the space where the primary copy was first created (see (6.3.26)).

We say that files with an unlimited lifetime are of ***File Storage Type*** PERMANENT. Only the owner of a file can remove the file (and its associated SURL) with PERMANENT File Storage Type. For limited lifetime, the File Storage Type *defines the behavior of the Storage Element when the life of the files expires*. In particular, VOLATILE files are limited lifetime files that the system can automatically remove when the lifetime expires. DURABLE files are limited lifetime files that cannot be automatically deleted by the system, but an error condition is raised instead and the user is notified.

Even though the default Storage Type for a file in an SRM is VOLATILE, in WLCG it has been decided that the only File Storage Type supported for the beginning of LHC operations is PERMANENT. This means that the associated SURL can only be removed by an explicit SRM call issued by the owner of the file.

A file has also associated Retention Quality and Access Latency. The possible values are the same as for spaces. The ***Retention Quality*** of a file describes the retention quality assigned to the file in the storage system at the moment when the master is written/created into the desired destination in the storage system. Files inherit the retention policy of the space where they are first created. For what concerns the ***Access Latency***, a file may have the same or slower access latency than the space that holds their instances.

The ***File Locality*** indicates if the file has an instance in ONLINE space or if it is NEARLINE or if copies exist in both ONLINE and NEARLINE space, if the file is LOST because of hardware failure or if it is currently UNAVAILABLE because of a temporary problem, or if its size is 0 (NULL).

In Figure 6.1 shows the association between the file and the space represents the relationship between a file and its primary or master space.

Depending on the particular type of operation that needs to be executed on a data set, an application first requests the SRM service to reserve a space with certain retention quality and access latency attributes and negotiates with the SRM the size and lifetime of the space. Once the space is reserved, a space token is returned. Using this space token, the application can request the SRM to create the master/primary copy for a set of files and to assign SURLs for them. Then it proceeds with transferring or producing the corresponding data to the space, using the negotiated transfer/access file protocols and the TURLs provided by the system. When the space is released (removed), the copies of the files (and the files) are treated accordingly to their File Storage Types: if file storage types are permanent, the files are kept in a default space until the owner explicitly removes the file entries from the SRM namespace; if file storage types are durable, the system performs the necessary actions at the end of their lifetime (it can alert the owner about the expired lifetime, but the files are kept in the default space); if file storage types are volatile, the system releases those files at the end of their lifetime and possibly removes the last copy and the SURL entries from the namespace.

In the meantime, other applications can ask the SRM to make copies of the files available for further processing. They can be retrieved in spaces with other characteristics but compatible with the file attributes, depending on the type of processing required by the application. Therefore, a file can have many ***space tokens*** associated to it at a given time.

SRM allows users to perform operations on an instance of a file. Such operations might not return a TURL, but might change the ***state*** of a file. For instance, a client might requests that an instance of a file stored on NEARLINE space is brought in ONLINE space for processing that will start at a certain deferred time. Therefore, at a given time files might have copies and/or TURLs associated. The UML state diagram for the file classes is shown in Figures 6.4 and 6.5.

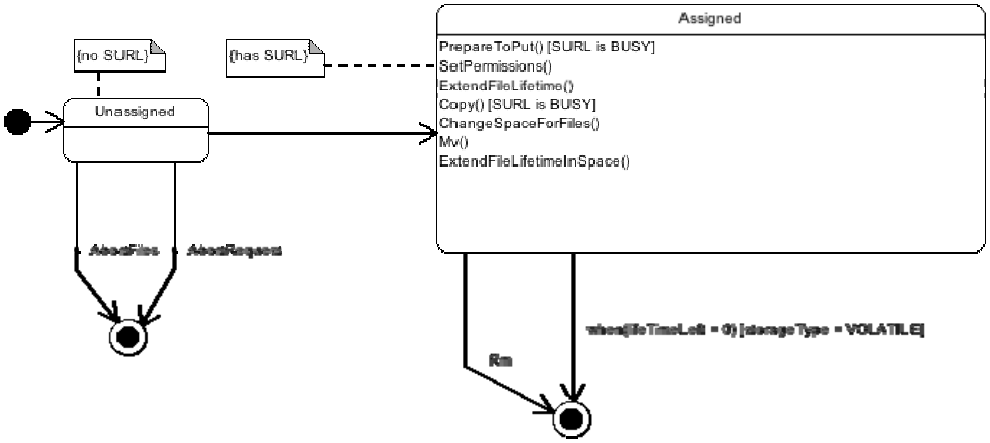


Figure. 6.4 The SRM v2.2 UML File class state diagram

Figure 6.5 shows that a file can be created via an *srmCopy* or an *srmPrepareToPut* operation. At this time a SURL has still not been assigned by the system to the file. However, the request can be aborted. Once the SURL is assigned, a file can be removed either by an explicit *srmRm* operation issued by the owner, by a privileged user (permanent or durable file), or by the system when the lifetime of the file expires.

After an *srmPrepareToPut* operation has been executed but before an *srmPutDone* is issued, the status of the file is BUSY (SRM_FILE_BUSY). This is also the status returned while a copy operation is in progress. While a file is busy, only *srmLs*, *srmSetPermission* operations are allowed. Operations like *srmSetPermission*, *srmMv*, and *srmExtendLifeTime* change the attributes of a file and modify only its internal status.

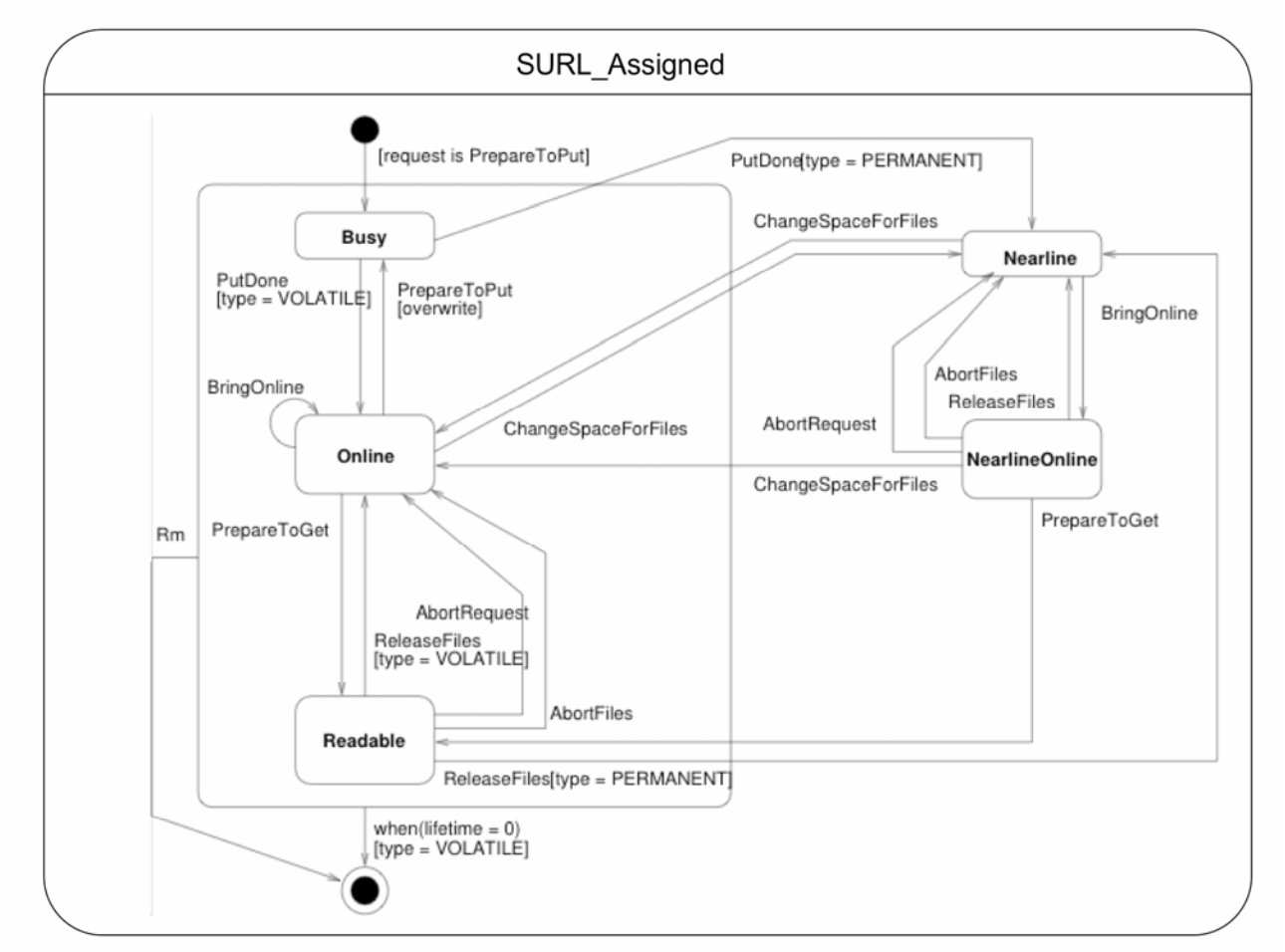


Figure. 6.5 The SRM v2.2 UML SURL_Assigned File state diagram

Figure 6.5 expands the possible states internal to the superstate "SURL_Assigned" and makes explicit the interactions between client and server while defining the possible states the server has to accommodate for the file class. As shown in the picture, the SRM interface is quite complex in this respect and could be simplified in terms of interactions between the client and server. The study that has produced the definition of the space and file state diagram has allowed the SRM-WG to identify incompleteness of the proposed interface and therefore to better specify it, in order to submit a proposal to OGF for both SRM version 2.2 and 3.0. In particular, a list of about 60 issues has been published in [121].

The state diagram for the copy and handle classes is much simpler and is shown in Figures 6.6 and 6.7.

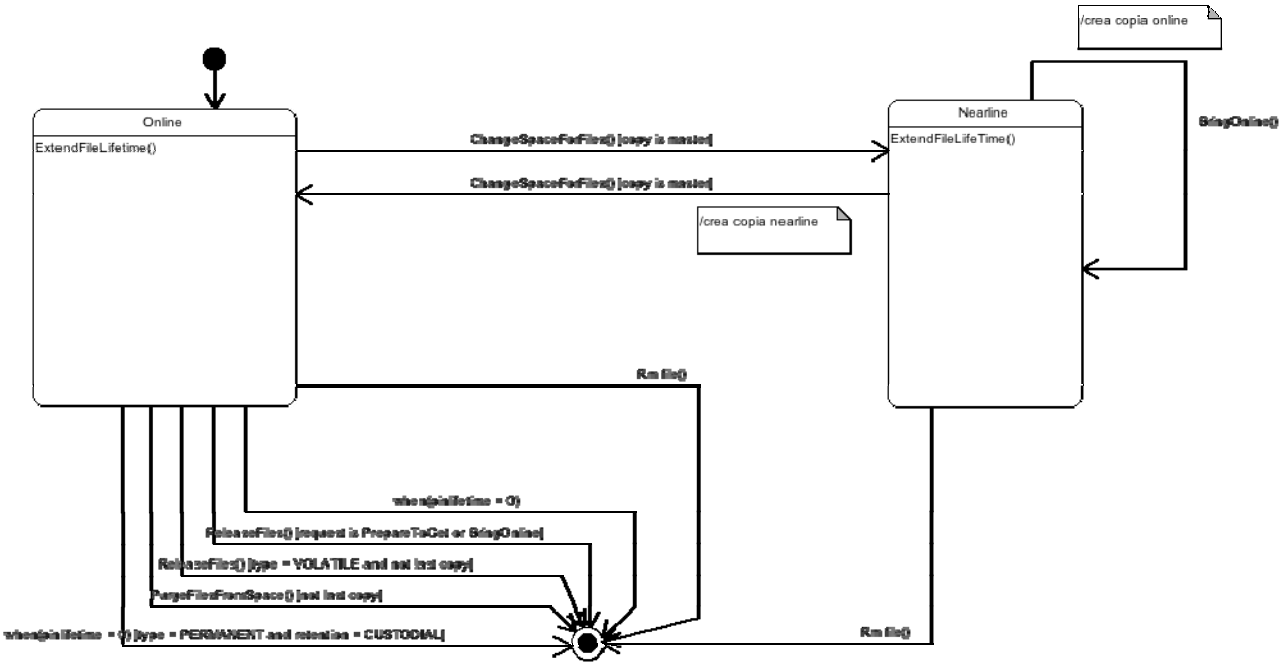


Figure. 6.6 The SRM v2.2 UML Copy class state diagram

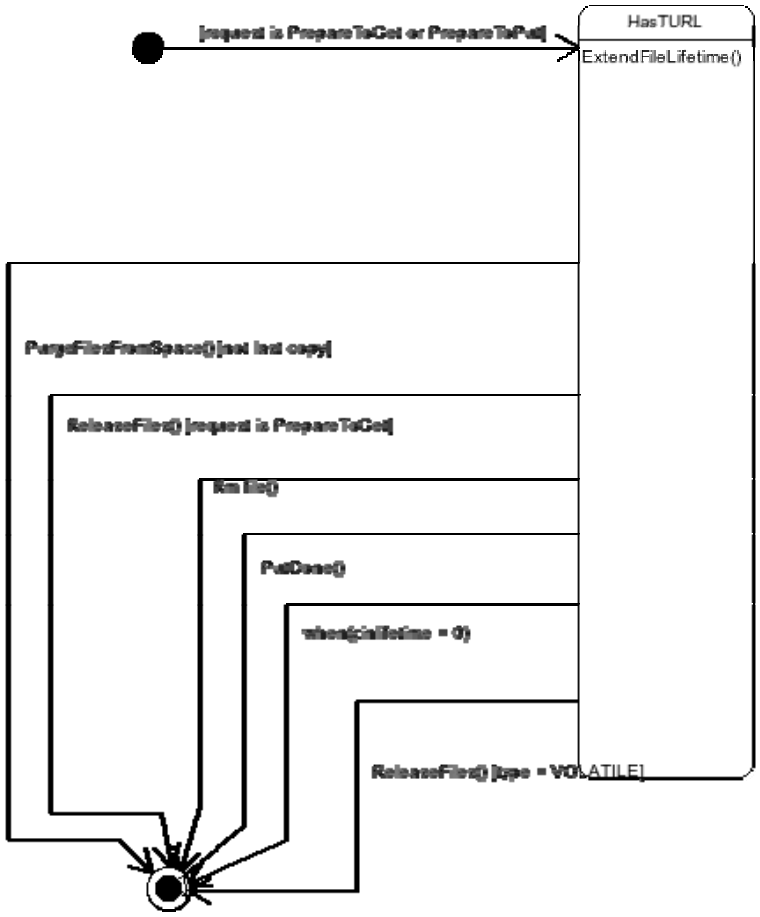


Figure. 6.7 The SRM v2.2 UML Handle class state diagram

Together with class and state diagrams, also UML activities diagrams have been drawn for each of the most complicated SRM data transfer functions in order to understand the allowed ordered sequence of actions between client and server. For instance, in Figure 6.8 shows the activity diagram for the *srmPrepareToPut* method. We can notice that when the status of the file is BUSY (after an *srmPrepareToPut* but before the *srmPutDone*) only a set of operations are allowed:

- The pin lifetime of the TURL can be extended, if necessary, in order to avoid the pin to expire while the write operation is taking place. The pin lifetime can be extended at any time during the write operation.
- The permissions on the file can be changed with *srmSetPermission*.
- The space where the file is being created can be released, if needed. However, if the file being created was declared permanent, the write operation continues and the *srmPutDone* is successful. The file will be then moved in a default space with the given space characteristics declared at file creation. If the file is volatile, the write operation complete successfully, however, the *srmPutDone* operation will fail with the error SRM_INVALID_PATH.
- Any other operation on the SURL or TURL will not be allowed and should return either SRM_FILE_BUSY or SRM_INVALID_PATH or SRM_FAILURE.

This study has helped identifying the sequence of these operations, discuss some of the implementation issues and agree on an unambiguous behavior as it emerges from [121].

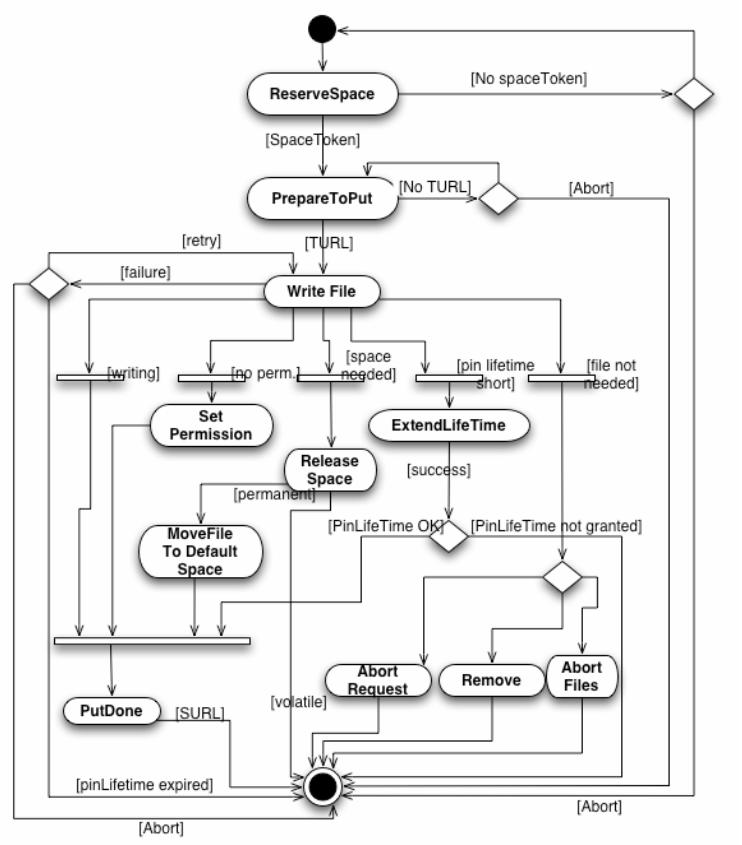


Figure. 6.8 The SRM File creation use case activity UML diagram

INFORMATION MODEL FOR THE SRM

In order to allow other Grid middleware services and client applications to contact the SRM endpoint for a given Storage Element, it is necessary to publish configuration and status information in the Grid Information System. To do so, an adequate model that is able to describe the different qualities of space and the Storage Classes requested by the WLCG experiments had to be put in place. After collecting possible use-cases coming from experiment applications and high-level middleware clients and services, we propose the following model for a Storage Element and the GLUE schema definition. Details of the proposed GLUE schema for a Storage Element for version 1.3 of the schema are given in [89].

7.1 The Storage Element

The following definition for a Storage Element has been agreed among the storage resource providers and developers in terms of GLUE schema.

A ***Storage Element*** *is an aggregate of Grid resources and services that allow Grid users to store and manage files together with the space assigned to them.*

Examples of storage elements are:

- The CASTOR system at CERN with its SRM interface and file access services, the physical storage backend being the set of robotic tape libraries and the pool of disk servers in front of them offering online/cache storage
- The LCG DPM system with its SRM interface and its set of disk servers, each of them managing given filesystems
- The dCache system with its SRM interface able to manage disk space and to interface to tape systems such as TSM, HPSS, Enstore.
- The StoRM system, offering an SRM interface to parallel filesystems such as GPFS
- A file server running a GriFTP daemon on the set of filesystems it manages

In terms of the GLUE schema, a Storage Element is characterized by properties such as:

- A ***Globally Unique Identifier*** that univocally identifies the SE.
- The ***Implementation***, the software system used to manage the storage devices and servers. Examples of this can be: CASTOR, dCache, DPM, StoRM, etc.
- The ***ImplementationVersion***. Through the version specific features of the SE can be exposed.
- The ***OnlineSizeTotal*** and the ***NearlineSizeTotal.*** This is the sum of the Total Size of Online and Nearline nominal space. For instance, the NearlineSizeTotal reports the total nominal space on tape not considering compression. These sizes are reported in GB (10^9 bytes).

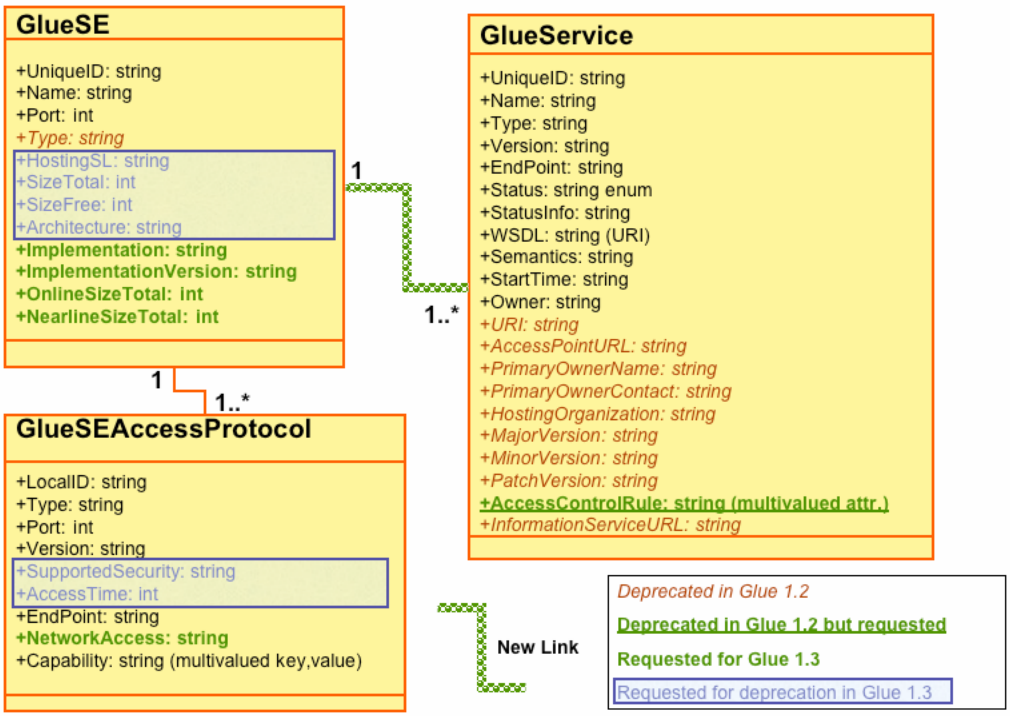


Figure. 7.1 The SE description in the GLUE schema

Figure 7.1 shows the proposed schema for GLUE v1.3 for a Storage Element. The classes shown exist already in GLUE v1.2. The (blue) boxes highlight the attributes that we proposed to deprecate in GLUE v1.3; the (red) italic attributes are already deprecated in GLUE v1.2. The bold (green) attributes are the ones we proposed to introduce, while the (green) bold underlined are the ones that are deprecated in GLUE v1.2 and we propose to keep in GLUE v1.3. Use-cases have been provided that express the need for the proposed changes.

The following considerations have been made in order to propose the SE schema:

1. It was felt that a GlueSE should not publish a size since it is not easy to come up with a meaningful and coherent definition of a size in case of very complex systems such as those managing robotic and online devices. It is better to move the size property to lower level descriptions of the storage system as detailed later. However, it seems feasible to expose Online and Nearline Total Sizes that can be used when making reports and obtained with simple queries that do not require looping through the GLUE schema when querying the information system.
2. The Storage Element Control protocol is described in the GlueService class (as defined by the GLUE schema v1.2 [88]) that describes the service. For instance, the SRM is one control protocol for the Storage Element. Storage Element proprietary or internal protocols can also be exposed in the information system. In this case, a GlueService object will describe the service associated to them.
3. The Service Access Control Rule needs to be kept in order to describe which VOs have access to the Storage Element service. This attribute was deprecated in GLUE v1.2.

7.2 The Access Protocol

The ***Access Protocol*** *describes how files can be accessed on a Storage Element.*

Figure 7.1 shows the proposed description for the GlueSEAccessProtocol. This Class has links to the GlueSE class as well as the GlueStorageArea class introduced later.

An Access Protocol is described by a ***Type*** and a ***Version*** (e.g. gsiftp 1, gsiftp 2, dcap 1.7.6, etc.)
The list of Access Protocol Types is defined already in GLUE schema v1.2 [88].

The ***Capability*** entry is used to define further Access protocol properties for which there is no explicit entry in the GLUE schema. However, it has been noted that a generic entry such as capability is hard to query since it is expressed via a string that can have a free format and might vary from site to site. For instance, the Capability field can be used to specify restrictions such as WAN read-only/LAN read-write.

The ***NetworkAccess*** attribute in the Access Protocol can be used to specify if the protocol is valid for a LAN or a WAN

Before the introduction of the SRM interface, only the so-called "classic" Storage Elements were supported in WLCG. These were disk servers, serving their volumes to the computing farm via the NFS protocol and running a GridFTP daemon to satisfy Grid transfer requests. The "file" protocol was therefore published in the information system to announce that standard POSIX file access was supported. However, such a configuration for an SE turned out to be unmanageable since many of the requirements needed in a Grid environment could not be accommodated. As a consequence, the "file" protocol was dropped. With the introduction of the SRM, implementations such as StoRM provide an SRM interface to parallel filesystems (GPFS) and therefore allow for standard POSIX file access. Therefore, the protocol "file" that does not appear in the list of SEAccessProtocolType for GLUE v1.2 needs to be reintroduced. The "file" protocol will also be used to describe all those proprietary protocols that allow for native POSIX I/O.

Many WLCG experiments (ALICE and LHCb) have requested to be able to use directly the xrootd protocol to access their data file. Also this protocol is missing in the list of SEAccessProtocolType. In general, it is suggested that the list of protocols appearing in the SEAccessProtocolType is approved by IANA in order to allow for the standardization of names used.

The GLUE class CESEBind was used for "classic" SE to publish how a volume exported by the SE via NFS was mounted on a CE and how files could be accessed. We propose to reuse this class to publish as well a list of access protocols that can be used from the particular CE bound to that SE and which properties they expose. CESEBind might describe CE and SE that are not in the same domain. An example is given by the NIKHEF/SARA connection described in Chapter 6. In general, we assume that WNs are always in the same domain as a CE and that the CE describes also the characteristics of WNs in its own domain (there are no examples to the contrary). Therefore, a protocol published in the CESEBind is enabled also from all WNs connected to the specific CE. Nothing can be said for protocols not explicitly mentioned in a CE-SE binding.

If an SE can be accessed directly using one of the supported access protocols, then the SE should be described as a GLUE service using that access protocol as control protocol.

7.2.1 The Access Protocol Use Cases

In what follows we list a set of use cases that justify the proposed GLUE schema for the Access Protocol class.

1. A user who wants to find all CEs that can access the input data available on an SE. At the same time, the user's job will produce output data that need to be stored on an SE with enough space. The SE must support the same protocols that the user's job uses. This is a typical scenario for a user using JDL requirements with the gLite WMS or the LCG Resource Broker. The proposed schema allows for such a query.

2. It should be possible to distinguish between protocols that are allowed for read-only operations on WAN and protocols that can only be used by a subset of the supported VOs. This can be expressed through the Capability attribute in the GlueSEAccessProtocol object. However, it is clear that a better description of such capabilities need to be provided.

3. The GLUE schema needs to be backward compatible and describe as well old "classic" SEs. Therefore, in this case the GridFTP protocol should be considered a control protocol as well as an access protocol. The GLUE service associated to a "classic SE" reports the GridFTP as control protocol. In fact, operations such as mkdir, rmdir are normally done by the control protocol. On the "classic" SE such operations are performed through GridFTP. Therefore this protocol should be published as both control and access protocol for a "classic" SE.

7.3 The Storage Component

A Storage Component refers to a physical partition of the storage space. *A **Storage Component** identifies a specific storage with certain properties*. They are the following:

1. Retention Policies: CUSTODIAL or REPLICA or OUTPUT
2. Access Latency: NEARLINE or ONLINE (or OFFLINE)
3. Access Protocols (examples: rfio, dcap, file, etc.)

The concept of a Storage Component was introduced to describe the type of storage that is used to offer a certain quality of storage. For instance a Storage Component is a tape set (a set of tape dedicated to store files logically grouped) or a pool of filesystems.

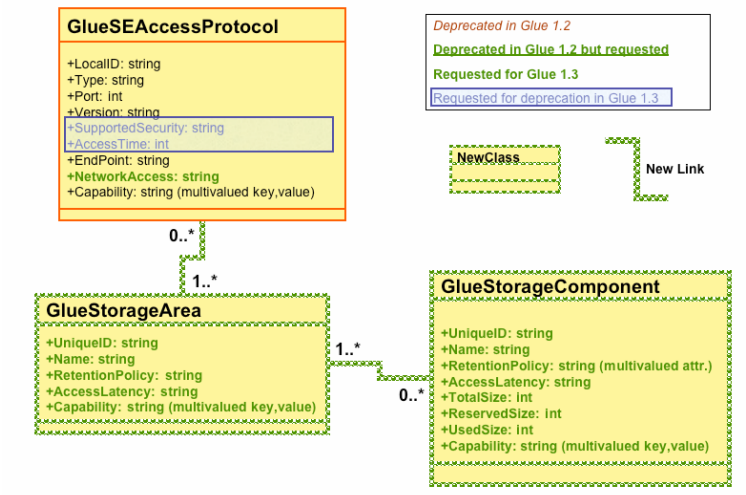


Figure. 7.2 The Storage Component in the GLUE schema

Figure 7.2 shows a UML class diagram for the GlueStorageComponent class and its relations with the GlueStorageArea class, introduced later.

The Storage Component implements a multi-valued ***Retention Policy*** (the ones supported) and ***Access Latency***. The Storage Component publishes Total, Reserved and Used Size. They are defined as follows:

Total Size: *is the total space size that this Storage Component can provide expressed in Gigabytes. It is the nominal capacity of the Storage Component subsystem (tape, dvd, disk, etc.)*

Reserved Size: *is the size of space reserved but not yet used, expressed in Gigabytes.*

Used Size: *is the size occupied by files that are not candidates for garbage collection.*

A Storage Component can be optionally identified by a ***Name***. This can refer for instance to the name of the DPM pool of filesystems composing the Storage Component.

Let us now make some considerations about the Storage Component in order to better map the GLUE class to existing entities in the storage service implementations:

1. The WLCG MoU for the implementation of SRM v2.2 states that the only retention policies supported in SRM v2.2 are CUSTODIAL and REPLICA.
2. Two Storage Components cannot overlap in order not to double count space.
3. In case of tapes the Total Size reports the nominal capacity of the tape (i.e. without considering the nominal compression factor). LHC experiments data are normally compressed and the compression algorithm used by most tape drives does not provide any benefit. Normally LHC experiments compress and uncompress data on the fly. Other VOs might take advantage of the compression algorithm used by tape drives. It is up to them to take this factor into account when counting up the space published by Storage Components.
4. A tape-only storage component might still have some disks in front for performance reasons. Such disks should be totally hidden and not published in the information system.
5. We feel that quotas should be specified at the level of a Storage Component. However, at this time we would like to defer the matter of thinking through how to express and/or implement quotas in the GLUE schema, as a tentative exploration showed that there are non-trivial issues to be dealt with.

7.3.1 The Storage Component Use Cases

A user can query the information system to get a description of each single component type in a Storage Element. Therefore, it is possible to find out the total amount of space on tape and/or disk summing up the published total size for all Storage Componets part of a Storage Element.

The Storage Component represents the storage space the LHC experiments are willing to pay for. Any extra storage needed for optimizing system performance (for instance a specialized cache space layer between a tape system and an online disk system) should be hidden and not exposed via the information system.

7.4 The Storage Area

The Storage Area is a logical partition of the total available space.

A ***Storage Area*** *defines a portion of the total available space that can span different kinds of storage devices within a Storage Element and in case of WLCG it implements a Storage Class instance.*

A Storage Element can have multiple Storage Areas. In principle any combination of Storage Components in a Storage Area is possible. For instance, a Storage Area can include a certain set of tapes and the space served by several disk servers. However, two Storage Areas can share the same Storage Components, but implement different policies.

Several Virtual Organizations can share the same Storage Area and can reserve space in it. This is for instance the case for the default Storage Area in WLCG.

A VO might have multiple Storage Areas reserved to it. SRM spaces are reserved in Storage Areas. Through the *srmReserveSpace* SRM call, an application passes the ***User Space Token Description*** and gets back a ***Space Token***, as we have seen in Chapter 6. For put operations, the middleware (GFAL, lcg-utils, FTS, etc.) takes the token description as input, converts it to a space token, which is passed to the underlying SRM method. Therefore, a User Space Token Description might have multiple Space Tokens associated to it in a single SRM. In fact, the same User Space Token Description can be used in separate requests to reserve space (statically and dynamically). The SRM has a call (*srmGetSpaceTokens*) that lists all the space tokens associated with a particular Token Description. The client can loop over them and pick a token that has the desired properties, e.g. the space associated is not marked as full by a metadata query result. In conclusion, a User Space Token Description is a tag given by the user to identify the set of spaces reserved through multiple storage reservation requests.

For WLCG only the space in a Storage Area reserved by Virtual Organization Managers and identified per VO by a ***User Space Token Description*** is optionally published in the Information System. In the same SE, there could be many Storage Areas that implement the same Storage Class characteristics.

Figure 7.3 represents a Storage Element, a Storage Area shared among a few Virtual Organizations, and User Space Token Descriptions for space reserved by those Virtual Organizations.

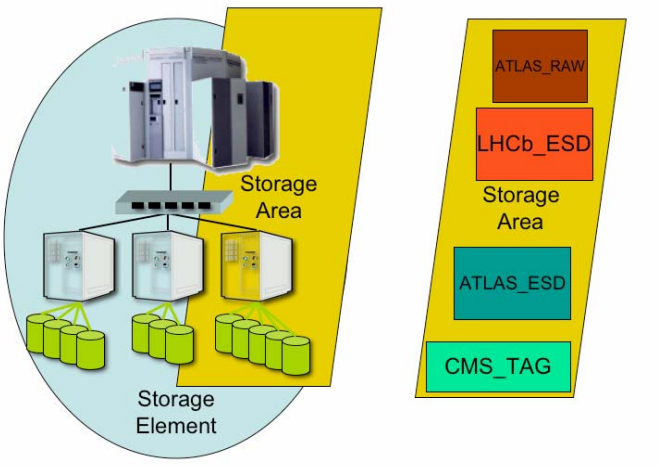


Figure. 7.3 Storage Area shared among ATLAS, LHCb and CMS, and User Space Token Descriptions for ATLAS RAW, LHCb ESD, and CMS TAG data. The Storage Area is shared among three VOs.

It has to be noted that in WLCG, dynamic user space reservation is optional. Furthermore, at least initially clients should not use dynamic space reservation. Therefore, for shared Storage Areas it is

impossible to guarantee a given amount of space to a VO. The situation is depicted in Figure 7.4. Therefore, in WLCG Storage Areas are normally dedicated to a single VO.

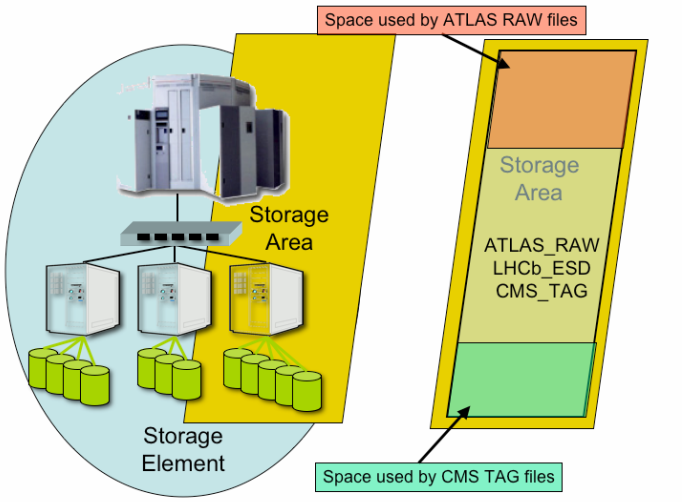


Figure. 7.4 Storage Area shared between VOs in a system with no dynamic space reservation functionalities. The space for LHCb data gets smaller because of the space occupied by ATLAS and CMS data.

Figure 7.5 represents a Storage Area allocated to only one VO, its Storage Components, the User Space Token Description and the Space Token returned by a dynamic SRM Reserve Space operation (possibly generally available with SRM v2.3).

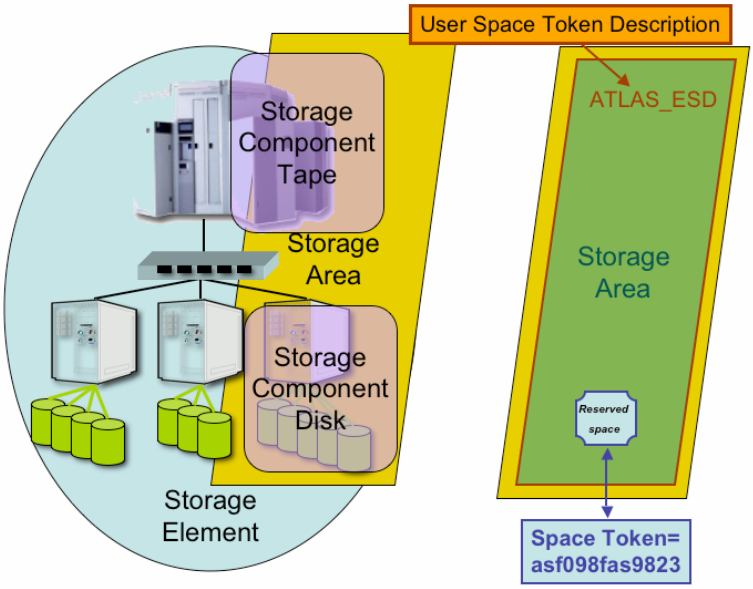


Figure. 7.5 Storage components and Space Tokens

In case a Space Token is not specified for an SRM Put operation, the system will use the default Storage Area that is generally shared between multiple VOs.

In the Glue Schema, a ***Name*** optionally identifies the Storage Area. This can be an internal identifier the site administrator uses to refer to that Storage Area. The Name can be published in the Information System.

Beside its name, a Storage Area has associated multi-valued ***Capability*** that can vary from implementation to implementation.

Retention Policy and ***Access Latency*** are also properties of a Storage Area and contribute to define the Storage Class associated to this Storage Area. The SRM v2.2 specification defines the possible values for Retention Policy and Access Latency. The SRM v2.2 WLCG MoU specifies the possible values for WLCG (the ones that define the Storage Classes specified in Chapter 3).

Figure 7.6 describes the GlueStorageArea and GlueVoStorageAreaAssociation classes. These classes do not exist in GLUE v1.2.

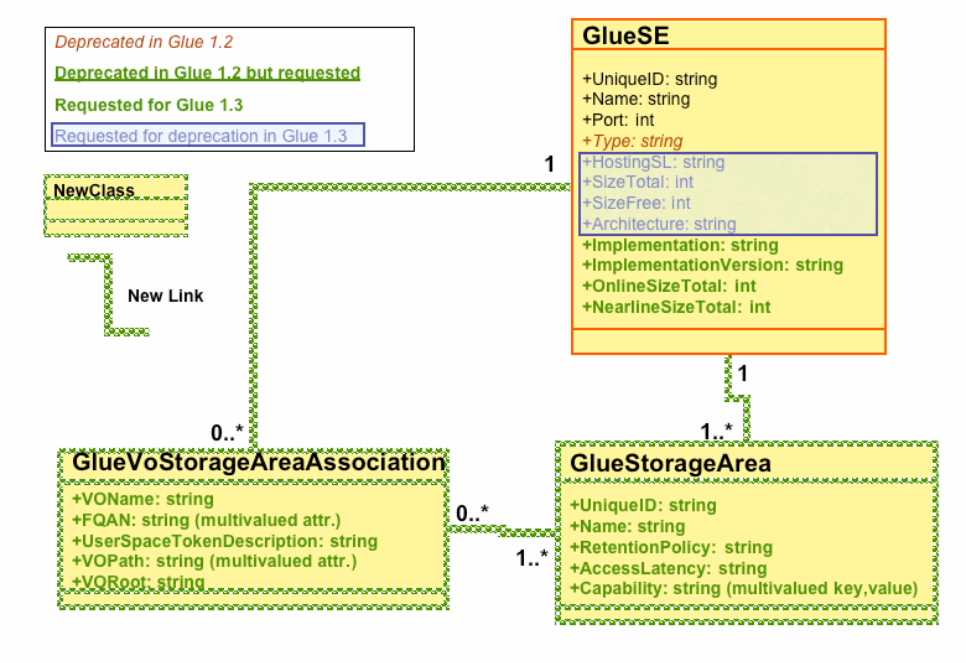


Figure. 7.6 The Storage Area in the GLUE schema

Storage Areas can be used to match all storage services that can provide a given class of Storage to a VO FQAN. In this case, the User Space Token Description does not need to be known or published. This could be the case for non LHC VOs.

7.5 The VO-Storage Area Association

For statically allocated space and for space reserved by administrators, the VO ***User Space Token Description*** is published in the Information System as part of the ***VoStorageAreaAssociation*** GLUE class, in order to allow a VO to find out per SE the space and the space characteristics reserved to that VO.

The ***VoStorageAreaAssociation*** GLUE class allows for the association between User Space Token Description and a Storage Areas. The ***VO Name*** describes which VO has access to this Storage Area. A

VOMS FQAN can be published as well to describe which users (with specific privileges) within the VO actually have access to that Storage Area. The VOName attribute in GlueVoStorageAreaAssociation cannot be avoided since we cannot assume that the VO name can be derived from the FQAN.

Optionally, a multi-valued ***VOPath*** associated with a VO specific Storage Area can be published. The paths can be used to build VO SURLs for files, and we assume that they are protocol independent. VOPath is mandatory when referring to classic Storage Elements to find out VO specific GridFTP directories on the server. For "classic" Storage Elements, the ***VORoot*** subdirectory published for the "file" protocol per VO, indicates the VO specific path that needs to be appended to the CEAccessPoint published in the CESEBind class to have the full VO Path for direct file access on the WN.

Storage Areas might have zero or more VoStorageAreaAssociation objects linked to them. This is to accommodate the need to publish available storage not yet assigned to VOs.

7.6 The Storage Paths

In SRM v2.2 the file namespace is orthogonal to the Storage Class, which can be derived from the Space Token supplied when the file is stored. In practice, however, there may be reasons to couple the name space with certain aspects of the quality of storage. dCache and CASTOR allow users to specify in which tape set a file should reside. dCache implements such a feature through the so-called Storage Groups. Storage Groups are associated to directory paths. Therefore in dCache a path may not only identify a Storage Class but also many other properties such as the tape set the file should reside on (for instance, real experimental data coming from the detector should reside on a set of tapes that must be different than the set of tapes dedicated to simulated data). StoRM as well implements Storage Classes through paths: at the moment a Space Token is not enough to describe the Storage Class.

The WLCG experiments have decided to structure the directory namespace to map it appropriately to Storage Classes. In this case, all sites would have to agree to do the same mappings. The user space token description needs to be passed anyway as input to the SRM Put/Copy calls. The path can also determine the tape set associated to the files since this has many advantages for managing and locating data.

The name space structure ought not to be necessary, as dCache can infer the Storage Group from the space token as well in the near future, and also the StoRM developers are trying to implement a characterization of the space to be used via the space token only.

7.7 Free and available space

It has been discussed if space is an attribute that needs to be published in the GLUE schema in other storage related classes beside the Storage Component. It was felt that only at the Storage Component level one could differentiate among the different kinds of storage available and avoid double counting of storage capacity assigned.

As regards the used space, it has been noted that such information might be inaccurate and not useful.

In WLCG tape space is considered to be infinite. For what concerns disk cache space we can note that ***Used space*** can be of three types:

1. Space used by files
2. Space allocated by space reservation methods. Part of this space is potentially available to put files.

3. Space which is currently used by files being migrated to tape but will be available as soon as the migration is over.

We felt that it was better to define used space as the size occupied by valid files (not candidates for garbage collection) and to differentiate between used size and reserved size, as the space reserved but not yet used. The free space can be inferred from the two values published, but in general it cannot be taken as available to store more files (there can be ACLs, quotas, … that could make such attempt fail).

As a further consideration we have to notice that the Information System can only give a snapshot of the Grid status at a given time. Therefore it could well be that a request for space reservation might fail even if the information system shows that space is available at a given time.

7.8 The Storage Component GLUE class

Storage Components describe low-level details of the storage devices part of a Storage Area. In principle, the storage unit exposed to applications is the Storage Area. Therefore the entire Storage Component class is optional and can remain unpublished for very complex systems. Also, it is not clear what the use-cases justify the need to expose the Storage Components in the information system. Figure 7.7 represents this situation.

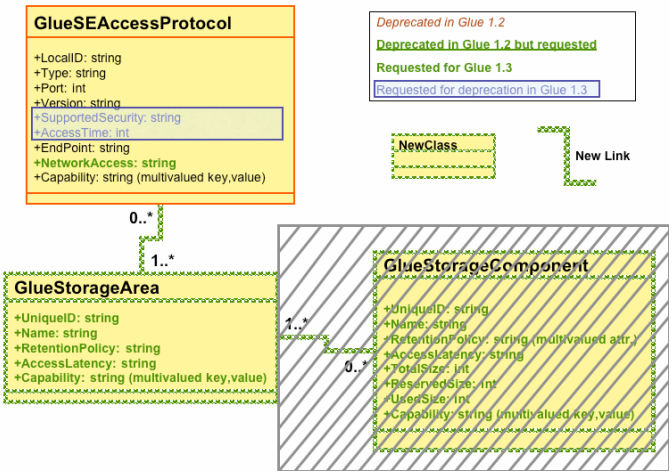


Figure. 7.7 The Storage Component Class is optional

A summary UML diagram of the proposed GLUE schema for the Storage Element for SRM 2.2 is given in Figure 7.8.

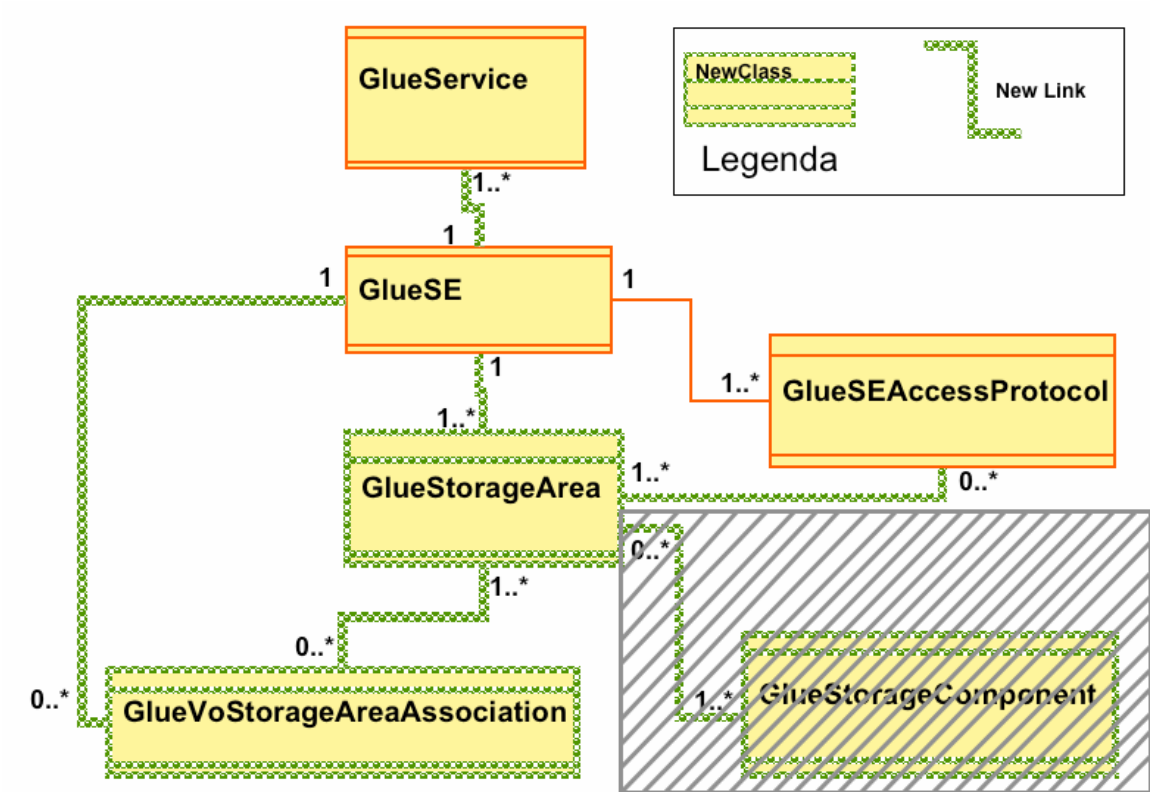


Figure. 7.8 The Storage Service description in GLUE v1.3

Chapter 8

TEST AND EVALUATION OF SRM IMPLEMENTATIONS

The completeness and coherence of the SRM protocol has been verified using the analysis made in Chapter 6, which allowed us to define a model for the proposed SRM interface. In particular, state and activity diagrams have helped identify many protocol interactions not initially foreseen by the specifications, as defined in [82]. Another important aspect in the definition of a protocol and in checking its efficiency is the verification against a few implementations. The verification process helps understand if foreseen transactions make sense in the real world. It shows as well if the protocol adapts naturally and efficiently to existing storage solutions. Furthermore, this process allows us to set up a test bed where real use cases can be tested. In what follows we describe the design of a functional test suite to verify the compliance of the implementations with the defined interface and their interoperability. In particular, an analysis of the complexity of the proposed SRM interface shows that a high number of tests need to be executed in order to fully check the compliance of the implementations to the specifications. Therefore an appropriate testing strategy has to be adopted in order to reduce the number of tests to be performed. A specific language, the S2, has been adopted for a fast implementation of test cases.

8.1 Theory of Testing

Testing activities aim at finding differences between the actual and the intended behavior of a system. In particular, [93] gives the following definition:

Testing *is the process of executing a program with the intent of finding errors.*

In fact, the process of testing is done in order to improve the implementation of a system and make it perform as expected, as close as possible to the user requirements and to the system specifications.

In [93], the author analyzes a set of principles that should be respected when executing tests. He gives guidelines and suggests strategies for identifying a good testing suite. One of the basic principles is to accurately identify the input data and describe the correct expected output for each of the possible inputs.

We assume that the intended behavior of a system is described by a ***specification***. More precisely, a specification describes all acceptable behaviors of the system. Obviously, we assume that any behavior (i.e., any particular sequence of stimuli and responses) of the system can be modeled in terms of the semantics of the language used for the specification. Then, if S is the set of all possible behaviors of a system, we say that S ***satisfies*** a specification SP if all the statements of SP are true when applied to S.

For example, let us assume that SP is composed of statements about pairs of the form `<x,y>`, where `x` is an input value and `y` is the corresponding computed output. S is then the set of all pairs that can be computed by the system. If SP contains the single statement "`for all x>0: y>0`", then S satisfies SP if and only if all pairs `<x,y>` in S with `x>0` have `y>0`.

In symbols:

$$S \models SP$$

specifies that a system S satisfies (has the same semantic as) the specification SP.

An ***Oracle O*** is a decision procedure that decides whether a test was successful or not.

Let us use the symbol $\models_O$ to indicate that the system S reacts according to the test set T_{SP} as observed by an Oracle O:

$$S \models_O T_{SP}$$

We say that a test set is ***exhaustive*** if and only if the following formula holds:

$$(S \models SP) \Leftrightarrow (S \models_O T_{SP})$$

In order for a test to be effective it should be ***exhaustive***, i.e. it must describe fully the expected semantics of the specifications, including valid and invalid behaviors. If we indicate with `input` an input value and with `output` the corresponding possible computed output of a system, we can formalize the ***exhaustive set of tests*** for a given specification as follows [94]:

```
(8.1.1) T_Exhaustive = {<behavior,result>| behavior ∈ {<input,output>},
                        result=true if behavior represents a valid behavior,
                        result=false if behavior models an invalid behavior}
```

A test set T_{SP} is ***pertinent*** if and only if it is:

- ***Valid***: no incorrect system behaviors are accepted
- ***Unbiased***: no correct system behaviors are rejected.

However, exhaustive test sets are not practicable in the real world since they imply infinite testing time. Furthermore, it is possible to identify an unbiased set of tests, but we cannot guarantee that a test suite is valid. Several techniques are applied in order to reduce the number of test cases to be written, while conserving validity. One of these is to state hypotheses about the ***System Under Test*** (SUT) in order to make a correct abstraction of its behavior. This is done with the intent of making the SUT look simpler and therefore easier to test (***test modeling***) [95]. Only if all hypotheses correspond to a valid generalization of the behavior of the system the final test set will be pertinent (valid and unbiased).

Test modeling is particularly efficient when dealing with programs or services that aim at performing specific tasks, following user requirements.

8.1.1 The Black Box methodology

In order to verify the compliance to a protocol of a specific implementation a test-case-design methodology known as ***Black Box testing*** is often used. The Black Box testing technique focuses on identifying the subset of all possible test cases with the highest probability of detecting the most errors. In particular, the most popular black box testing approaches are ***Equivalence partitioning***, ***Boundary-value analysis***, ***Cause-effect graphing*** and ***Error guessing*** [93]. Each of these approaches covers certain cases and conditions but they do not ensure the identification of an exhaustive testing suite. However, one of them can be more effective than the other ones depending on the nature of the SUT. Furthermore, the latter approach is the one that normally increases the number of test cases to be considered, based on the experience acquired on the SUT.

8.1.1.1 Equivalence partitioning

The equivalence partitioning black box approach is based on two fundamental assumptions. A good test is one that has the following properties:

- It reduces the number of test cases that must be written in order to achieve a reasonable result
- It covers a large number of input conditions

In order to achieve the two goals mentioned above, the equivalence partitioning methodology focuses on identifying a set of input arguments or conditions that are representative for a large number of conditions that give the same results (i.e. equivalent inputs). The combinations of input argument values are grouped in classes, ***equivalence classes***, so that a test of a representative value of each class is equivalent (i.e. it produces the same result) to a test of any other value. Therefore, if one test case in an equivalence class detects an error, all other test cases in the same equivalence class would be expected to find the same error. Consequently, input conditions are partitioned into two or more groups: for instance, valid and invalid input to the system. The same approach can be applied to output conditions. A test set is then written focusing on the partitions that cover most of the cases.

As an example, let us consider an integer input argument that can assume values greater or equal to zero but less or equal than 2 ($0 \leq n \leq 2$, $n \in \mathbf{N_0}$). We can then identify two equivalence classes:

$$\text{Equivalence class 1} = \{ n \in N_0 \mid n \leq 2 \}$$
$$\text{Equivalence class 2} = \{ i \in Z \mid i < 0,\ i > 2 \}$$

We can then identify the 2 test cases where the input argument is equal to 1 or -1. The test case where the input argument is equal to 1 represents the equivalence class 1 while the test case where the input argument assumes value -1 represents equivalence class 2.

More generally, if the behavior of the SUT depends on N input parameters, an equivalence class for the SUT is a maximal set of N-tuples such that the behaviors of the SUT for each member of the class is equivalent under a given criterion.

The equivalence partitioning methodology has been applied to write the ***basic SRM test suite***, as described later on in this chapter.

8.1.1.2 Boundary-value analysis

Boundary-value analysis concentrates on boundary conditions. Therefore, one or more elements are selected so that each ***edge*** of the equivalence class is the objective of a test.

*The **edge** of an equivalence class E is defined as the set of N-tuples of E whose components have limit values that the corresponding parameters can assume.*

In this case the ***result*** space is as important as the input space. In other words, one should keep into account the values of output parameters as well as other possible output conditions or limitations imposed by the requirements on the SUT.

Let us continue with the example in Section 8.1.1.1. In order to select our test cases, we would choose the values 0 and 2 for the equivalence class 1 and -1 and 3 for the equivalence class 2. The system could behave differently depending on the value of the input argument. If the *expected* behavior (output condition) of the system is the same for both values 0 and 2, then we consider these two cases equivalent and we only write a test case for one of them. However, if the expected behavior is different, then we have to write a test case for each condition. For instance, if the value 0 for the input argument means that the system has to use an internal default value while 2 is a legal value, then we have to test for both cases.

Boundary-value analysis has been applied to design the ***use-case SRM test suite***, as described later.

8.1.1.3 Cause-effect graphing

One important aspect in the verification of a protocol is to understand the effect of the ***combinations of input parameters*** as well as ***events*** of any type. For instance, in the analysis of a system we have to consider the concurrency of actions that can affect the state of a system, the combination of input arguments that are correlated, the history of events that were triggered on the system before the specific request is made, etc. If the number of optional input parameters or the set of methods that can affect the state of a system is high, then the task of defining a set of pertinent tests can be rather difficult. This is for instance the case for the SRM specifications.

Cause-effect graphing helps in selecting, in a systematic way, a set of meaningful test cases. It has the beneficial side effect of pointing out incompleteness and ambiguities in a system specification. A cause-effect graph is a formal language into which the semantic content of the specification is expressed. The ***cause*** is a given a set of input conditions, while the ***effect*** is an output condition or a system transformation. A cause-effect graph is a Boolean graph linking causes end effects. In Figure 8.1, the basic cause-effect graph symbols are shown.

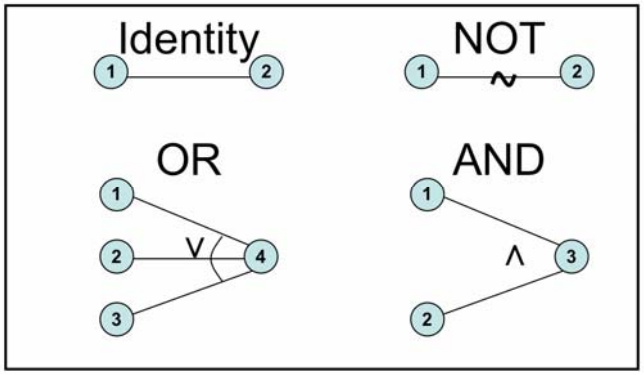


Figure 8.1: Cause-effect graph symbols. On the left of each symbol there are the causes while on the right there are the effects. Standard logic is used to derive the effect from the cause condition(s).

The graph can be converted into a decision table, tracing state conditions in the graph. Assuming that a given effect is true, the graph is traced back for all combination of causes that will set that effect to true. For each combination, the state of all other effects is determined and placed in a column of the table. The columns of the table represent test cases.

Let us consider a function `f(x,y,v,w)` with 2 input arguments `x` and `y`, and 2 output arguments `v` and `w`:

```
f: X x Y -> V x W
```

where `X`, `Y`, `X`, and `W` are the set of values taken by `x`, `y`, `v`, and `w` repectively.
The conditions expressed in the specifications for the function `f` are:

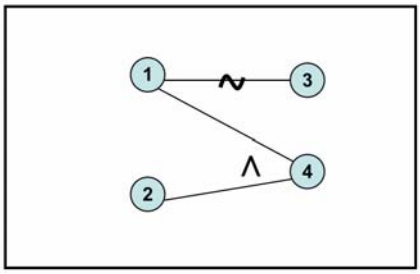
a) $X=\{0,1\}$
b) $Y=N_0$
c) $x=1 \text{ and } y=1 \Rightarrow v=1$
d) $x=0 \Leftrightarrow w=0$

From these specifications we can derive the following causes and effects:

Causes:
1. x=1
2. y=1

Effects:
3. w=0
4. v=1

The conditions a)-d) are expressed by the following cause-effect graph:



We now generate a limited-entry decision table assuming first effect 3 to be true. If effect 3 is true, then cause 1 is false, which means that effect 4 is false, no matter what is the status of cause 2. If instead effect 3 is false, then cause 1 is true, which means that either cause 2 is true, in which case effect 4 is also true, or cause 2 is false which means that effect 4 is also false. The decision table is shown.

	a	**b**	**c**
1	0	1	1
2		0	1
3	1	0	0
4	0	0	1

In the rows we have the causes and effects and we have one column for each test case to be written (a,b,c).

8.1.1.4 Error guessing

The error guessing approach does not follow any specific rule, but it is totally based on intuition and experience. Once a specific set of tests runs on a system, it is rather natural for the person analyzing the results to understand the weakness of an implementation and therefore to design and exercise specific test cases.

8.2 The SRM case

Let us analyze the case of one of the popular SRM methods in order to understand how complex the task can be to exhaustively test such a function. In particular, let us estimate the number of test cases that we have to write in order to fully test one of the SRM methods.

The ***srmReserveSpace*** function is defined as follows:

srmReserveSpace

```
IN:
   string               authorizationID,
   string               userSpaceTokenDescription,
   TRetentionPolicyInfo retentionPolicyInfo,
   unsigned long        desiredSizeOfTotalSpace,
   unsigned long        desiredSizeOfGuaranteedSpace,
   int                  desiredLifetimeOfReservedSpace,
   unsigned long []     arrayOfExpectedFileSizes,
   TExtraInfo[]         storageSystemInfo,
   TTransferParameters  transferParameters

OUT:
   TReturnStatus          returnStatus,
   string                 requestToken,
   int                    estimatedProcessingTime,
   TRetentionPolicyInfo   retentionPolicyInfo,
   unsigned long          sizeOfTotalReservedSpace,
   unsigned long          sizeOfGuaranteedReservedSpace,
   int                    lifetimeOfReservedSpace,
   string                 spaceToken
```

where under **IN** we list the input parameters and under **OUT** the output returned values. The underlined parameters are mandatory while the others are optional. The meaning of the parameters has been introduced in Chapter 6.

In order to verify that this method has been correctly implemented, we can proceed in steps. Let us suppose that we want to restrict ourselves to analyzing the set of input arguments and test an implementation for the correct set of return values. In particular, according to the ***equivalence partitioning*** approach let us count the number of meaningful test cases to consider in applying this technology.

Let us indicate with N_{arg} the total number of input arguments. We can assume that each argument can have values in the following classes:

1. Valid values
2. Invalid values with a correct format
3. Invalid values with an incorrect format
4. The NULL value (i.e., the argument is absent)

Furthermore, some arguments like *retentionPolicyInfo* or *transferParamenters* can assume a finite set of values that have all to be tested in order to check the response of the system against the specifications. A heuristic approach will tell us that the total number of tests to be executed to cover these cases is:

$$N_T = 3^{Narg} + 3\sum_i^K V_i + \prod_i^K V_i \qquad (8.2.1)$$

where N_T is the number of tests to be executed. The formula above has been derived as follows. The number 3 is obtained subtracting the class of valid values from the number of equivalence classes considered above (1-4). This is to avoid counting the N-tuple of values for the input arguments obtained using valid values for the arguments that can assume a finite set of possible values. N_{arg} is the total number of input arguments; K is the number of arguments that can assume a finite set of possible values; Vi is the number of values that argument i can assume. In case there are no arguments that can assume a countable finite set of values, we assume that

$$K=N_{arg} \text{ and } V_i=1 \ \forall i,\ i=1...N_{arg}$$

In the formula (8.2.1) the first term counts all tests obtained with the possible combinations of the arguments using the cases 2., 3., and 4. The second and third terms derive from the remaining combinations of possible values assumed by arguments with a finite set of values.

In our ***srmReserveSpace*** case, the function has nine input arguments (of which two are mandatory). The *retentionPolicyInfo* argument is a 2-tuple as defined by the formula (6.3.2). If we assume that only one of the elements of the 2-tuple can be NULL (i.e. either RQ or AL can be NULL) but not both at the same time, we can see that this argument can assume 11 possible fixed values. The same applies to *transferParamenters* that can assume 44 fixed values (see formulas (6.3.3) and (6.3.4)). Applying the formula (8.2.1) the number of possible test cases to be evaluated is *at least* 20.332.

After this, we still have uncovered test cases left: for instance those derived from the specification constrains. As an example, the ***srmReserveSpace*** specification states that the method can be asynchronous. In this case, the server must return a status such as SRM_REQUEST_QUEUED or SRM_REQUEST_INPROGRESS and a request token must be returned to the user. For these cases the ***cause-effect graph analysis*** can be used. However, the number of test cases that could be derived cannot be counted a priori, since it really depends on the specific method.

Furthermore, test cases that exploit the type of the input arguments or examine the interaction among different SRM functions are not counted.

The SRM v2.2 specification defines 39 methods with in average of six up to a maximum of ten input arguments. Therefore, considering the number of arguments that can assume a high number of fixed values, a test suite to cover just the equivalence partitioning approach for all of these functions would require more than 200.000 test cases. If we then consider input combinations, output evaluation, and concurrency of actions and their effects, the number of tests to be written becomes immediately unmanageable.

8.3 Reducing the test set: a use-case based analysis

In order to reduce the number of possible test cases to be written and executed to verify the compliance of an implementation to the standard, we made a few assumptions dictated by the requirements of the WLCG experiments, as described in Chapter 3. The goal is to keep our test set valid. Test cases have been obtained combining the black box methodologies for a fast convergence, as shown in what follows.

Let us analyze again the case of one of the most popular SRM functions: ***srmReserveSpace***.

We have already described the input and output parameters of the ***srmReserveSpace*** function. The behavior of the function is described in details in [82]. In order to write an efficient and pertinent test suite for this function, we started examining the LHC experiment requirements and to determine the set of input arguments that are requested by users and that most implementations do not ignore. In particular, as stated in Chapter 3, it is important for this function to be able to guarantee a space of a certain quality to users for the amount of time required. Therefore, important input parameters to consider are:

- *retentionPolicyInfo*
- *desiredSizeOfGuaranteedSpace*
- *desiredLifetimeOfReservedSpace*.

As described in Chapter 3, there are only three possible combinations allowed for *retentionPolicyInfo*, given the Storage Classes that need to be supported in WLCG:

- Tape1-Disk0=(CUSTODIAL, NEARLINE),
- Tape1-Disk1=(CUSTODIAL, ONLINE),
- Tape0-Disk1=(REPLICA, ONLINE).

To this, we have to add possible system defaults: (CUSTODIAL, NULL) or (REPLICA, NULL). Furthermore, if we consider the data transfer and access patterns described in Chapter 3 and the importance of transfer tasks to distribute data from Tier-0 to Tier-1 and Tier-2 centers, it is very unlikely that input and output buffers and specific protocols on servers managing them are left to the users to define and allocate during an ***srmReserveSpace*** call. Normally, centers will statically configure space buffers with predetermined characteristics (such as protocols, connectivity type, etc. – see Chapter 6) that applications can reserve. Therefore, if users request a certain combination of *retentionPolicyInfo* and *transferParameters* for which no space buffer has been allocated at a certain location, the SRM server will just ignore the *transferParamenters* value passed. The *authorizationID* is foreseen in order to specify a user identity and allow the system to grant permission for space reservation. However, in WLCG such a parameter is not used by any of the implementations taken into consideration since they are all VOMS aware. The role or group of a user is derived from his proxy. In WLCG only VO managers can reserve space and make it available to users of a VO. Therefore, the VOMS proxy is checked and appropriate privileges to reserve space are guaranteed. No other input parameter is relevant in order to guarantee the requested functionality.

Given the considerations above, there are only the following input conditions to consider:

1. *retentionPolicyInfo*
2. *desiredSizeOfGuaranteedSpace*

3. *desiredLifetimeOfReservedSpace*
4. *User proxy* (Note that the user proxy is not a direct input parameter but it is available to the server via the GSI enabled SOAP request)
5. *transferParameters*

We have already mentioned that *retentionPolicyInfo* can assume only five possible values. Therefore, applying the formula (8.2.1) in the case of the equivalence partitioning for only these arguments, the number of test cases to consider is 263.

In order to further reduce the tests to be performed, we consider combining two black box approaches: equivalence partitioning and boundary-value analysis.

We first apply the input equivalence partitioning methodology to the 263 cases in order to find still possible cases of equivalence. Then, we analyze the values that the output parameter *returnStatus* can assume and we reduce ourselves to significant cases that produce the same output or error conditions. Then, while selecting the values of input parameters, we limit ourselves to conditions close to the *edges* allowed by the specifications. We analyze the cases where a valid equivalence class was represented by a test case with an input condition just under the threshold allowed and an invalid equivalence class by a test case with an input condition just above the threshold allowed. This procedure should allow us to drastically reduce the number of test cases to keep into consideration.

In what follows we explain in detail the procedure. Let us consider the *retentionPolicyInfo* parameter with its five possible values. The StorageClass Tape0Disk1=(REPLICA, ONLINE) must be supported by all implementations and therefore it is a case to be considered. The classes Tape1Disk0=(CUSTODIAL, NEARLINE) or Tape1Disk1=(CUSTODIAL, ONLINE) are equivalent since the implementation can either support them or not. The cases (CUSTODIAL, NULL) or (REPLICA, NULL) are also equivalent since the server can either provide a default for the Access Latency or not. In the second case, the server must return SRM_INVALID_REQUEST. Taking these considerations into account, we can limit ourselves to consider only four cases:

1. retentionPolicyInfo = (REPLICA, ONLINE)
2. retentionPolicyInfo = (CUSTODIAL, NEARLINE)
3. retentionPolicyInfo = (REPLICA, NULL)
4. retentionPolicyInfo = NULL

The input argument *desiredSizeOfGuaranteedSpace* is mandatory. Therefore, it cannot be NULL (invalid condition). Furthermore, this argument can only be greater than 0 and can be as big as allowed by the argument type (unsigned long). However, practically asking a system to reserve 150TB of disk space is a boundary condition, since very few systems have such a space available for reservation by a single user. On the other hand, a request of 10GB is rather reasonable.

The *desiredLifetimeOfReservedSpace* is an optional argument that can either assume a NULL or 0 value (equivalent to requesting the system default), or -1 for infinite lifetime, or a value greater than 0. However, in this last case, the system can successfully complete the call returning a lifetime shorter than requested by the user. This is perfectly in accordance with the specification since this parameter is meant to negotiate a possible lifetime with the server.

Let us now analyze the case of the *User Proxy*. Here, we can limit ourselves to only the following cases:

1. The user does not have a valid proxy

2. The user has a proxy that does not allow him/her to reserve space
3. The user has a valid proxy that allows him/her to reserve space

In case 1., the SRM server returns the status SRM_AUTHENTICATION_FAILURE. In case 2, the SRM server returns SRM_AUTHORIZATION_FAILURE. Case 3 represents instead a valid equivalence class.

For the *transferParameters* input parameter, a similar analysis can be done.

The result of the equivalence partitioning analysis performed for the ***srmReserveSpace*** method is given in Table 8.1.

Input condition	Valid Input Equivalence Classes	Invalid Input Equivalence Classes
retentionPolicyInfo	**(REPLICA, ONLINE) (REPLICA, NULL) (CUSTODIAL, NEARLINE)**	**NULL**
desiredSizeofGuaranteedSpace	**150TB, 10GB**	**0, NULL**
desiredLifetimeOfReservedSpace	**NULL, 0, >0, -1**	**-2**
Proxy	**Valid**	**Invalid, No VOMS role or group**
transferParameters	**NULL,Valid**	**Invalid**

Table 8.1: Equivalence partitioning classes for ***srmReserveSpace***

The list of test cases that we designed in order to verify the behavior of the implementations when the ***srmReserveSpace*** method is called is given in Table 8.2. As shown, we have reduced the number of our test cases significantly while creating an unbiased test suite given our assumptions based on the requirements from the WLCG experiments.

N.	Test case and input arguments	Value of returned arguments: status code

1	retentionPolicyInfo=(CUSTODIAL, NEARLINE);
	desiredSizeofGuaranteedSpace=10GB;
	desiredLifetimeOfReservedSpace=3600sec;
	Valid proxy;
	transferParameter=NULL
		SRM_NOT_SUPPORTED |
		SRM_NO_FREE_SPACE |
		SRM_NO_USER_SPACE |
		SRM_LOWER_SPACE_GRANTED (spaceToken) |
		SRM_SUCCESS (spaceToken) |
		SRM_REQUEST_QUEUED (requestToken) |
		SRM_REQUEST_INPROGRESS (requestToken)

2	retentionPolicyInfo=(REPLICA, NULL);
	desiredSizeofGuaranteedSpace=10GB;
	desiredLifetimeOfReservedSpace=3600sec;
	Valid proxy;
	transferParameter=NULL
		SRM_INVALID_REQUEST |
		SRM_NO_FREE_SPACE |
		SRM_NO_USER_SPACE |
		SRM_LOWER_SPACE_GRANTED (spaceToken) |
		SRM_SUCCESS (spaceToken) |
		SRM_REQUEST_QUEUED (requestToken) |
		SRM_REQUEST_INPROGRESS (requestToken)

3	retentionPolicyInfo=NULL;
	desiredSizeofGuaranteedSpace=10GB;
	desiredLifetimeOfReservedSpace=3600sec;
	Valid proxy;
	transferParameter=NULL
		SRM_INVALID_REQUEST

4	retentionPolicyInfo=(REPLICA,ONLINE);
	desiredSizeofGuaranteedSpace=150TB;
	desiredLifetimeOfReservedSpace=3600sec;
	Valid proxy;
	transferParameter=NULL
		SRM_NO_FREE_SPACE|
		SRM_NO_USER_SPACE|
		SRM_LOWER_SPACE_GRANTED (spaceToken)|
		SRM_SUCCESS (spaceToken)|
		SRM_REQUEST_QUEUED (requestToken)|
		SRM_REQUEST_INPROGRESS (requestToken)

5	retentionPolicyInfo=(REPLICA,ONLINE);
	desiredSizeofGuaranteedSpace=NULL;
	desiredLifetimeOfReservedSpace=3600sec;
	Valid proxy;
	transferParameter=NULL
		SRM_INVALID_REQUEST

6	retentionPolicyInfo=(REPLICA,ONLINE);
	desiredSizeofGuaranteedSpace=-2;
	desiredLifetimeOfReservedSpace=3600sec;
	Valid proxy;
	transferParameter=NULL
		SRM_INVALID_REQUEST

7	retentionPolicyInfo=(REPLICA,ONLINE);
	desiredSizeofGuaranteedSpace=0;
	desiredLifetimeOfReservedSpace=3600sec;
	Valid proxy;
	transferParameter=NULL
		SRM_INVALID_REQUEST

8	retentionPolicyInfo=(REPLICA,ONLINE);
	desiredSizeofGuaranteedSpace=10GB;
	desiredLifetimeOfReservedSpace=NULL;
	Valid proxy;
	transferParameter=NULL
		SRM_INVALID_REQUEST|
		SRM_NO_FREE_SPACE|
		SRM_NO_USER_SPACE|
		SRM_LOWER_SPACE_GRANTED (spaceToken and lifetime=default)|
		SRM_SUCCESS (spaceToken and lifetime=default)|
		SRM_REQUEST_QUEUED (requestToken)|
		SRM_REQUEST_INPROGRESS (requestToken)

9	retentionPolicyInfo=(REPLICA,ONLINE);
	desiredSizeofGuaranteedSpace=10GB;
	desiredLifetimeOfReservedSpace=0;
	Valid proxy;
	transferParameter=NULL
		SRM_INVALID_REQUEST|
		SRM_NO_FREE_SPACE|
		SRM_NO_USER_SPACE|
		SRM_LOWER_SPACE_GRANTED (spaceToken and lifetime=default)|
		SRM_SUCCESS (spaceToken and lifetime=default)|
		SRM_REQUEST_QUEUED (requestToken)|
		SRM_REQUEST_INPROGRESS (requestToken)

10	retentionPolicyInfo=(REPLICA,ONLINE);
	desiredSizeofGuaranteedSpace=10GB;
	desiredLifetimeOfReservedSpace=-1;
	Valid proxy;
	transferParameter=NULL
		SRM_INVALID_REQUEST|
		SRM_NO_FREE_SPACE|
		SRM_NO_USER_SPACE|
		SRM_LOWER_SPACE_GRANTED (spaceToken and lifetime=infinite)|
		SRM_SUCCESS (spaceToken and lifetime=infinite)|
		SRM_REQUEST_QUEUED (requestToken)|
		SRM_REQUEST_INPROGRESS (requestToken)

11	retentionPolicyInfo=(REPLICA,ONLINE);
	desiredSizeofGuaranteedSpace=10GB;
	desiredLifetimeOfReservedSpace=-2;
	Valid proxy;
	transferParameter=NULL
		SRM_INVALID_REQUEST

12	retentionPolicyInfo=(REPLICA,ONLINE);
	desiredSizeofGuaranteedSpace=10GB;
	desiredLifetimeOfReservedSpace=3600sec;
	Invalid proxy;
	transferParameter=NULL
		SRM_AUTHENTICATION_FAILURE

13	retentionPolicyInfo=(REPLICA,ONLINE);
	desiredSizeofGuaranteedSpace=10GB;
	desiredLifetimeOfReservedSpace=3600sec;
	No valid VOMS group/role in proxy;
	transferParameter=NULL
		SRM_AUTHORIZAITON_FAILURE

14	retentionPolicyInfo=(REPLICA,ONLINE);
	desiredSizeofGuaranteedSpace=10GB;
	desiredLifetimeOfReservedSpace=3600sec;
	Valid proxy;
	transferParameter=Valid=
	(TRANSFER_MODE,WAN,gridftp)
		SRM_NO_FREE_SPACE|
		SRM_NO_USER_SPACE|
		SRM_LOWER_SPACE_GRANTED (spaceToken)|
		SRM_SUCCESS (spaceToken)|
		SRM_REQUEST_QUEUED (requestToken)|
		SRM_REQUEST_INPROGRESS (requestToken)

15	retentionPolicyInfo=(REPLICA,ONLINE);
	desiredSizeofGuaranteedSpace=10GB;
	desiredLifetimeOfReservedSpace=-2;
	Valid proxy;
	transferParameter=Invalid=
	(TRANSFER_MODE,WAN,pippo)
		SRM_INVALID_REQUEST

Table 8.2: List of test cases for ***srmReserveSpace***

With the equivalence partitioning and boundary condition analysis, we were able to identify the set of input and output conditions to validate the behavior of an SRM system. However, we still have not considered the restrictions imposed by the specifications and the semantics of the function to validate. For instance, the ***srmReserveSpace*** method can be asynchronous. In this case the system can queue the request, returning a request token to the application that has invoked it. This token can be used as input to the ***srmStatusOfReserveSpaceRequest*** method to check the status of the operation. The specification restrictions and the semantics of the method are performed using the ***cause-effect graphing*** methodology. In Figure 8.2 we list the causes and effects identified. In particular, the causes taken in consideration are numbered from 1 to 13, while the effects are numbered from 90 to 93.

1. retentionPolicyInfo is not NULL
2. retentionPolicyInfo is supported by server
3. Invalid input paramenters
4. desiredSizeOf GuaranteedSpace is not NULL
5. desiredSizeOfTotalSpace is not NULL
6. Default for desiredSizeOfTotalSpace is supported
7. desiredLifetimeOfReservedSpace is present
8. Default for desiredLifeTimeOfReservedSpace is supported
9. transferParameters is not NULL
10. transferParamenters is OK with retentionPolicyInfo
11. requestToken is returned
12. spaceToken is returned
13. sizeOfGuaranteedReservedSpace and lifetimeOfReservedSpace are returned

90. returnStatus is SRM_NOT_SUPPORTED
91. returnStatus is SRM_INVALID_REQUEST
92. returnStatus is SRM_REQUEST_QUEUED|SRM_REQUEST_INPROGRESS
93. returnStatus is SRM_SUCCESS|SRM_LOWER_SPACE_GRANTED

94. sizeOfGuaranteedReservedSpace=default
95. lifetimeOfReservedSpace=default
96. transferParameters is ignored

Figure 8.2: List of causes (1-13) and effects (90-93) for the ***srmReserveSpace*** method. The effects 94-96 are intermediate effects that need to be checked in test cases.

As it can be noted, the list of causes does not include only conditions on the input arguments, as it normally happens in cause-effect analysis. The *results* are also taken into considerations in order to represent the restrictions imposed by the protocol.

Figure 8.3 represents the analysis of the semantic content of the specification transformed into a Boolean graph linking the causes and the effects. This is the cause-effect graph for the ***srmReserveSpace*** function.

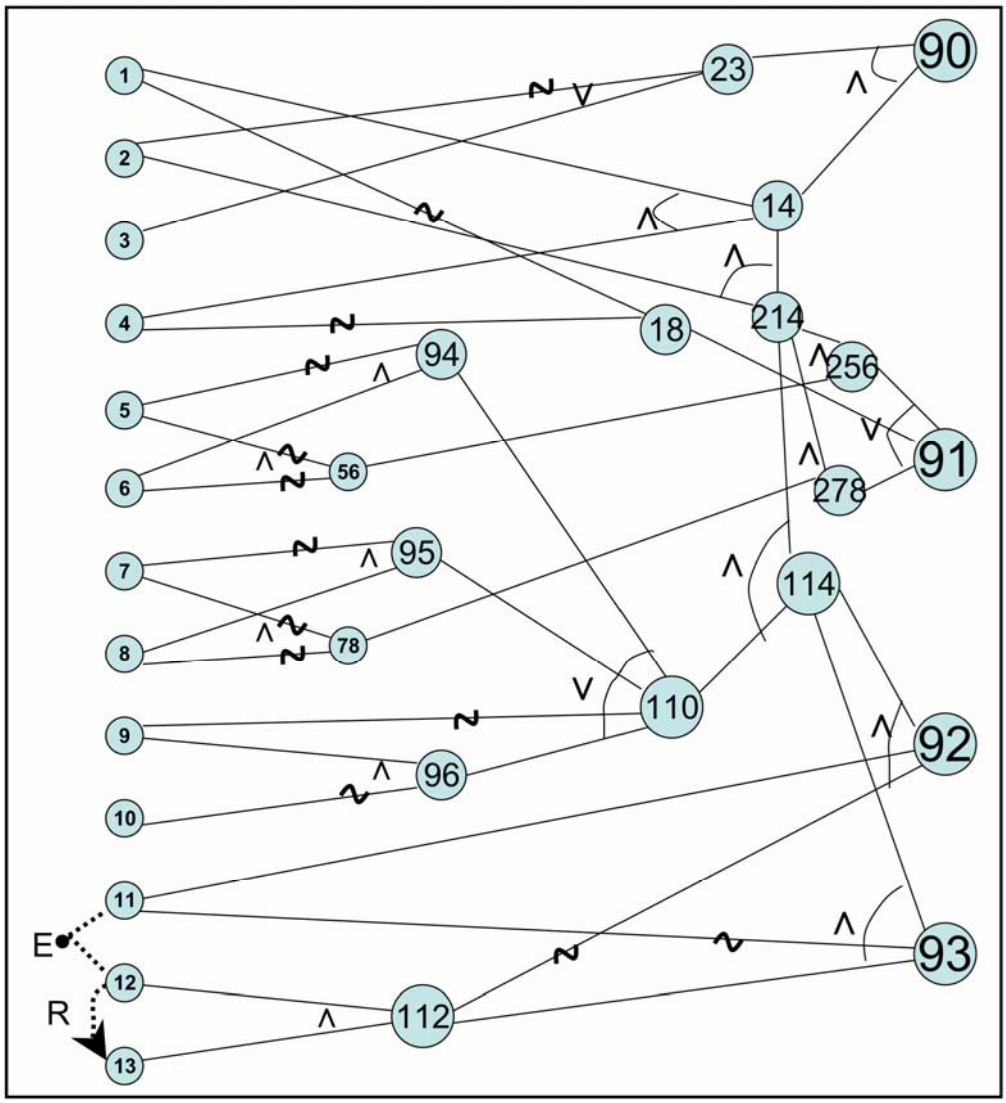


Figure 8.3 Cause-effect graph for the *srmReserveSpace* method

The intermediate nodes 56, 78, 23,14, 18, 110, 112, 114, 214, 256 and 278 have been created to facilitate the navigation of the graph. The nodes 94, 95, and 96 instead represent real effects foreseen by the specification that are somehow included in the final effects but need to be checked explicitly by a test program. For instance, if the ***desiredSizeOfTotalSpace*** is NULL (negation of cause 5) and the SRM system supports a default for ***desiredSizeOfTotalSpace*** (cause 6), then the return value ***sizeOfTotalSpace*** must be checked to be non-NULL and greater than 0. Causes 11, 12, and 13 are constrained by the ***E*** and ***R*** constrains: constrain ***E*** between nodes 11 and 12 express the fact that causes 11 and 12 cannot be simultaneously true; constrain ***R*** between nodes 12 and 13 represents the fact that if 12 is true then 13 must be true.

For ***srmReserveSpace*** the cause-effect graph has been translated into a limited-entry decision table, following the algorithm described in Section 8.1.1.3. Each column in the table represents a test case. Table 8.3 shows the decision table.

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
1	1	1	0		1	1	1	1	1	1	1	1	1	1	1	1
2	0				1	1	1	1	1	1	1	1	1	1	1	1
3		1														
4	1	1		0	1	1	1	1	1	1	1	1	1	1	1	1
5					0						0					0
6					0						1					1
7						0				0					0	
8						0				1					1	
9								1	0				1	0		
10								0					0			
11	0	0			0	0	1	1	1	1	1	0	0	0	0	0
12	0	0			0	0	0	0	0	0	0	1	1	1	1	1
13	0	0			0	0	0	0	0	0	0	1	1	1	1	1
90	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
91	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
92	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
93	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

Table 8.3: The decision table for cause-effect graph analysis for ***srmReserveSpace***. In the rows the causes and effects are reported. The columns represent the test cases.

The study above has allowed us to find inconsistencies in the specification. For instance, if the server did not support the *retentionPolicyInfo* value, both values for the *returnStatus* SRM_NOT_SUPPORTED and SRM_INVALID_REQUEST were considered correct by some implementations. Therefore, it was decided that the correct return code should be SRM_NOT_SUPPORTED. Furthermore, the specification only allowed for SRM_REQUEST_QUEUED to be returned when the method is asynchronous, while also SRM_REQUEST_INPROGRESS is possible for implementations that are fully parallel or threaded. Another example is the default lifetime for the reserved space. The specification set it to infinite or -1, while for practical implementation issues it was decided to leave the default to be an implementation choice.

In order to complete our functionality test on the ***srmReserveSpace*** method, we have also included some test cases derived from our model in order to study the behavior of the systems when concurrent calls were made interfering with each other. For instance, in the case of an asynchronous ***srmReserveSpace*** we studied the effects of an ***srmAbortRequest*** issued before ***srmReserveSpace*** is completed. We also checked the information returned by ***srmGetSpaceMetadata*** and the effect of an ***srmUpdateSpace*** once the ***srmReserveSpace*** is completed. The issues found are published in [96]. We have used the ***cause-effect graphing*** methodology also in this case, expressing in a formal way the SRM protocol in case of interacting functions.

Furthermore, the ***error guessing*** methodology was used in order to find recurrent errors due to the way a given system is designed. For instance, the dCache implementation for SRM v2.2 used to return generic error codes (SRM_FAILURE), even when the behavior was clearly defined by the specification for erroneous situations. This was due to a peculiar interpretation of the SRM specification. For instance, in the case of SRM transfer methods, the developers interpreted the file level return status as a file characteristic that changed with the file state. In the specification, the file level return status is always connected to the request issued and can be different from request to request, even if the file state does

not change. Special use cases were designed to discover specific situations. For instance, in the case of CASTOR a synchronization problem between the SRM server and the backend storage system was found. If an ***srmPrepareToPut*** operation was performed on a SURL after an ***srmRm*** on the same SURL, an ***srmStatusOfPutRequest*** returned first a failure and then a success on the same request. This was due to the fact that the remove operation was executed by the backend asynchronously while for the SRM this operation is synchronous. The SRM server was implemented not to consider this difference keeping into account the history of operations performed by the client on a specific SURL, before proceeding with satisfying further request on the same SURL.

A total of 52 tests have been written to test the compliance to the specification for ***srmReserveSpace***.

8.4 Experimental results

The analysis described has been performed over the 39 SRM methods foreseen by the specification to design a test suite to validate the correctness of the implementations with respect to the protocol established. We proceeded dividing the test suite in families that probed the SUTs with an increasing level of detail. In particular, we have designed and developed four families of tests:

1. **Availability**: the ***srmPing*** function and a full put cycle for a file is exercised (***srmPrepareToPut***, ***srmStatusOfPutRequest***, file transfer, ***srmPutDone***, ***srmPrepareToGet***).
2. **Basic**: the equivalence partitioning and boundary condition analysis is applied
3. **Use cases**: cause-effect graphing, exceptions, functions interference, and use cases extracted from the middleware and users applications.
4. **Interoperability**: remote operations (servers acting as clients) and cross copy operations among several implementations.

The total number of written tests is 2132. Figure 8.4 shows the web pages made available to the developers to check the output of the tests and a detailed explanation of the problems found.

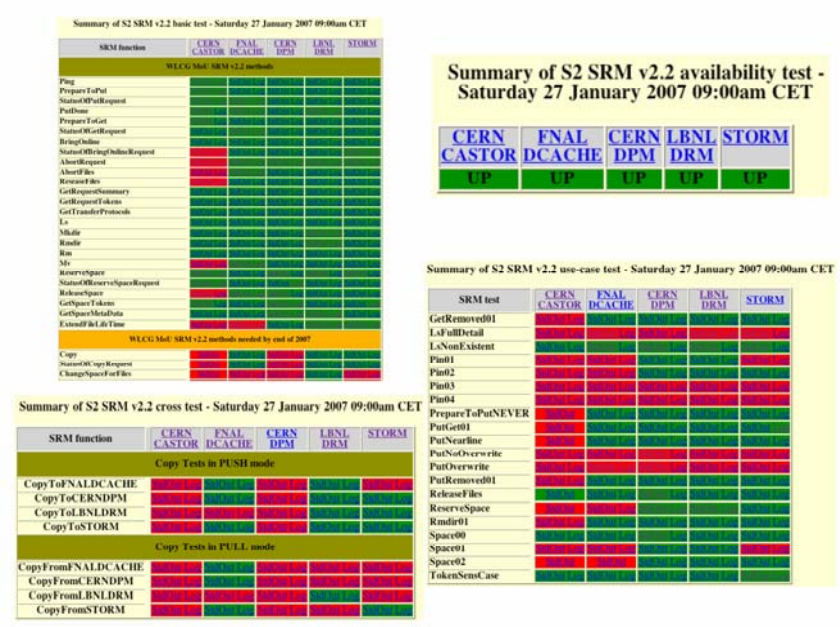


Figure 8.4: The web pages associated to the test results

To these families, we have also added other tests, the ***exhaust*** family, to exercise the format of the input arguments: strings too long, blank padded strings, strings padded with non-printable characters, malformed URLs, POSIX compliant paths with the maximum length allowed, etc. We have also written some ***stress*** cases that will be executed in a later point in time when suitable to the implementations.

The testbed that we set up includes five different implementations: CASTOR, dCache, LDPM, DRM, and StoRM. Two implementations have an MSS backend. These are CASTOR and dCache. The available implementations are located in four different countries: CASTOR and LDPM at CERN in Switzerland, dCache at FermiLab near Chicago in the U.S.A., DRM at Lawrence Berkeley National Laboratory near San Francisco in the U.S.A., and StoRM at CNAF in Bologna, Italy.

The tests are run automatically 6 times a day. The data of the last run together with the history of the results and their details are stored. Plots are produced every month on the entire period of run to track the improvements and detect possible problems.

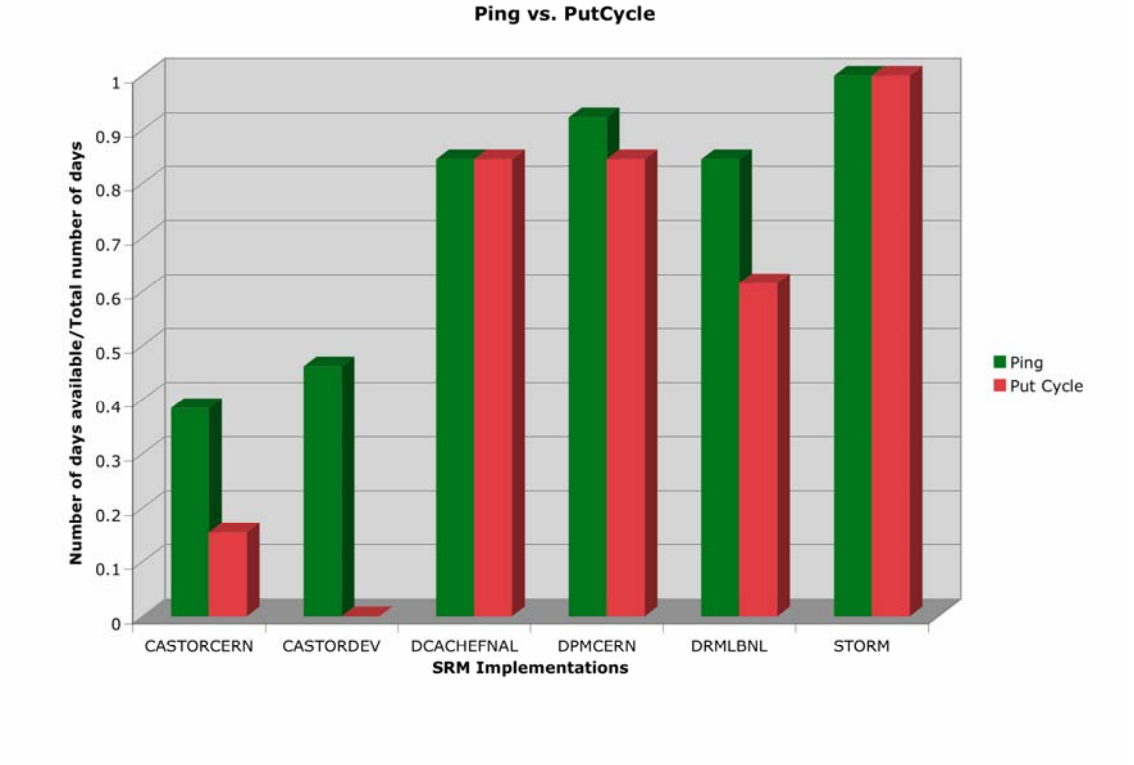


Figure 8.5: Results of the availability tests for 6 implementations. CASTORDEV is a development endpoint for CASTOR.

Figure 8.5 shows the plot of the availability results for all implementations. From the plot, it is possible to see that the CASTOR implementation was rather unstable. The instability problems have been identified in the GSI security plug-ins for gSOAP (a C++ implementation of SOAP), problems due to the Oracle database used in the backend, and some memory leaks identified in various part of the implementation.

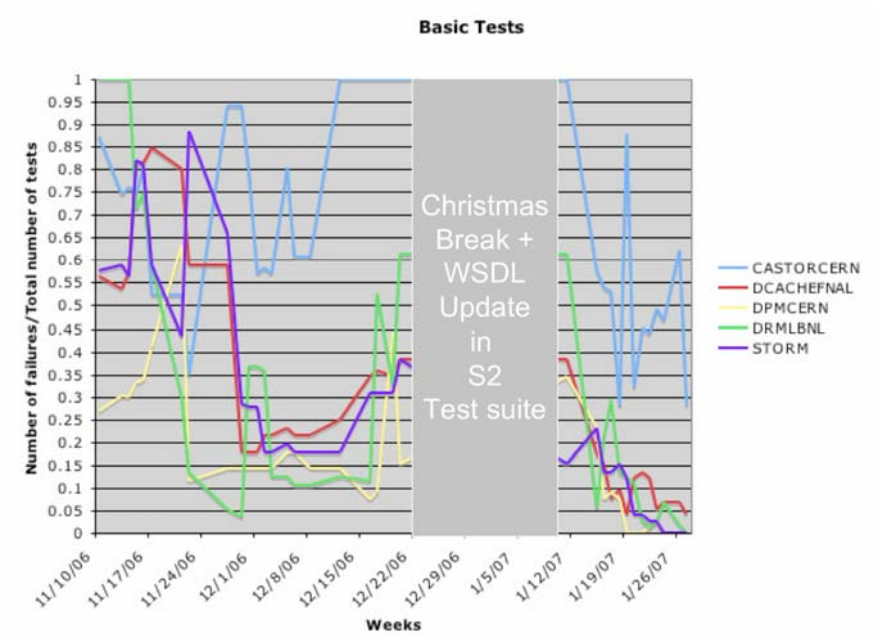


Figure 8.6: Basic tests: number of failures over total number of test executed over time for all SRM 2.2 implementations

Figure 8.6 plots the results of the basic tests for all implementations. In particular, the number of failures over the number of total tests executed is reported over time. It is possible to observe that after a month of instability, the implementations converged over a certain set of methods correctly implemented (around the 11th of December 2006). In the week of the 15th of December 2006, a new WSDL was introduced to reflect the decision made about the issues that the new SRM model and the analysis of the specification revealed. At the same time the developers introduced new features and bug fixes that produced oscillations in the plot. After the Christmas break and the upgrade of the testing suite to the new WSDL description, the implementations started to converge again toward a correct behavior with respect to the specifications. We can notice that toward the end of the testing period the number of failures is almost zero for all implementations. That is when testing should stop.

Figures 8.7 and 8.8 plots the results of the interoperability and use cases tests for all implementations. As it is done for the basic tests plots, the number of failures over the number of total tests executed is shown over time. It is possible to notice that the SRM implementations still show quite some level of instability. As far as interoperability is concerned, the copy operation can be executed in push or pull mode, as explained in Chapter 6. The copy tests exercise copy operations in both push and pull mode from all servers to all servers. From the plot we can see that the SRM 2.2 servers have implemented the copy mostly in one direction only.

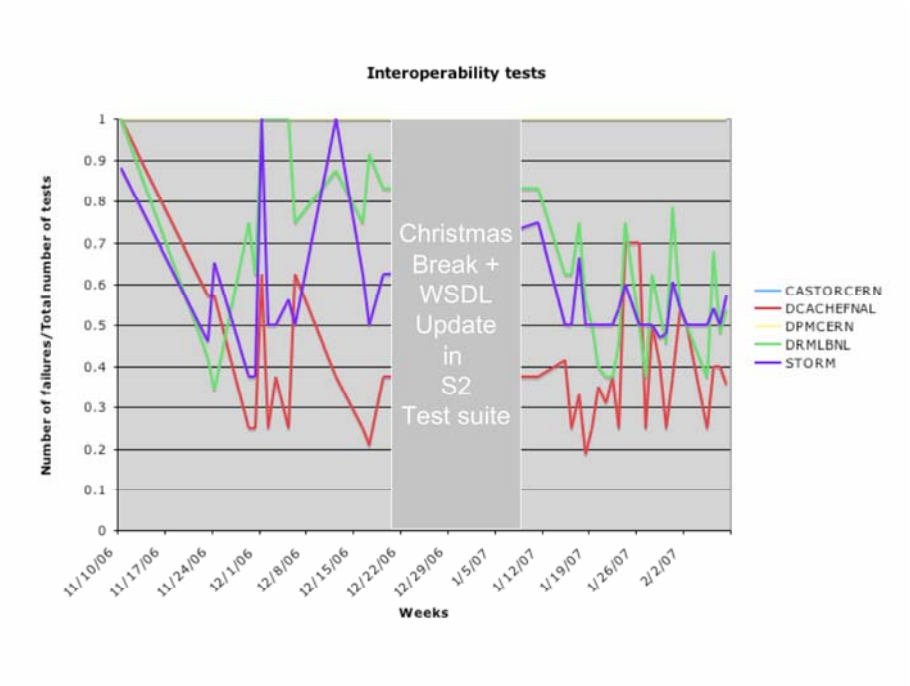


Figure 8.7: Interoperability tests: number of failures over total number of test executed over time for all SRM 2.2 implementations

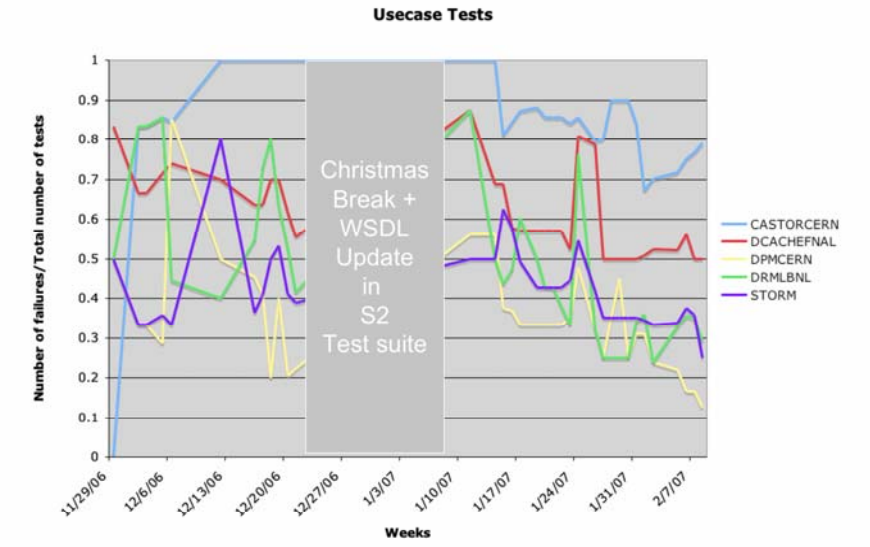


Figure 8.8: Use Case tests: number of failures over total number of test executed over time for all SRM 2.2 implementations

8.5 The S2 language

In order to speed up the development of the testing suite, we looked for a language with the following characteristics:

- It allows for the quick development of test programs that exercise one single test case.
- It helps minimize human errors that are typically made while writing test cases.
- It offers an easy way to plug-in external libraries such as an SRM client implementation.
- It offers a powerful engine for parsing the output of a test, expressing the pattern to match in a compact and fully descriptive way.
- It offers a testing framework that allows for parallel and sequential execution of testing units.
- It offers a "self-describing" logging facility that makes it possible to automatically publish the results of a test.

We identified such a language to be the S2 [97], a tree-based language with SRM 2.2 protocol support developed specifically for executing tests.

The S2 tree consists of ***branches***, which correspond to nodes in graph theory. The S2 interpreter starts the evaluation of an S2 tree at its root: the first branch with no indentation. Branches are in relationship with other branches (parents, children or disconnected). Together with a set of optional parameters, a specific ACTION defines the branch. The ACTION is expressed with clear syntax and semantics.

Actions have an ***execution value*** and an ***outcome***. The execution value is 0 for successful execution, or a positive integer related to the severity of failure. The outcome is a logical value, TRUE for success and FALSE for failure. It is possible to specify a threshold for the execution value under which execution has a TRUE outcome.

Actions can be composed by means of iterative structures (similar to for and while), parallel execution, AND-sequential execution, and OR-sequential execution. In the two latter forms of composition, execution of each branch depends on the outcome of the previously executed one: AND-sequential execution terminates as soon as a branch has a FALSE outcome, OR-sequential execution terminates as soon as a branch has a TRUE outcome.

One of the most powerful features of the S2 language is the possibility of using Perl compatible regular expressions as parameters of the ACTION where a response is expected. S2 also supports simple preprocessor directives.

The fundamental kind of action is the execution of an SRM command. The S2 interpreter recognizes a set of expressions with a command-line syntax that exercise all the implemented SRM calls. The outputs of an SRM command are stored in variables and, by use of regular expressions, they can be searched for the occurrence of given patterns. Other kinds of actions include tests involving the outputs of the SRM commands, and the execution of any operating system call.

Figure 8.9 shows a simple test example. In the example, the first branch is the execution of the SRM command srmLS. The output values go into the variables *requestToken*, *pathDetails*, and *returnStatus*, with the two fields explanation and *statusCode*. The regular expressions match the whole strings representing the output values.

The second branch, executed if the previous one is successful, makes some tests on the output values and outputs results with the echo system command. Finally, the last branch, executed if any of the previous ones fails, prints out an error message.

Various environment variables are used throughout the script.

```
% srmLs

srmLs $ENV{ENDPOINT} SURL[$ENV{SRM_ENDPOINT}] numOfLevels=0
requestToken=(?P<requestToken>.*) pathDetails=(?P<pathDetails>.*)
returnStatus.explanation=(?P<returnStatus_explanation>.*)
returnStatus.statusCode=(?P<returnStatus>.*)

&& TEST $MATCH{(SRM_SUCCESS|SRM_REQUEST_QUEUED|SRM_PARTIAL_SUCCESS|SRM_INPROGRESS) ${returnStatus}}
   SYSTEM echo "srmLs: OK: ${returnStatus_statusCode}" >> $ENV{S2_LOG}
   && TEST !$DEFINED{requestToken}
      SYSTEM echo "srmLs: Ls is synchronous" >> $ENV{S2_LOG}
   || SYSTEM echo "srmLs: Ls is asynchronous" >> $ENV{S2_LOG} && exit 0

|| SYSTEM echo "srmLs: KO: ${-returnStatus_statusCode} ${-returnStatus_explanation}" >> $ENV{S2_LOG} && exit ${!}}

0:srmLs srm://lxdpm01.cern.ch:8446 SURL[srm://lxdpm01.cern.ch:8446/dpm/cern.ch/home/dteam] numOfLevels=0
requestToken=(?P<requestToken>.*) pathDetails=(?P<pathDetails>.*)
returnStatus.explanation=(?P<returnStatus_explanation>.*)
returnStatus.statusCode=(?P<returnStatus_statusCode>.*)

path0=/dpm/cern.ch/home/dteam returnStatus.statusCode0=SRM_SUCCESS
size0=0 type0=DIRECTORY returnStatus.statusCode=SRM_SUCCESS

0:&& TEST "$MATCH{(SRM_SUCCESS|SRM_REQUEST_QUEUED|SRM_PARTIAL_SUCCESS|SRM_INPROGRESS) SRM_SUCCESS}"
0:   SYSTEM echo "srmLs: OK: SRM_SUCCESS" >> ./s2_logs/22DPMCERN/example1.log
0:   && TEST !$DEFINED{requestToken}
0:      SYSTEM echo "srmLs: Ls is synchronous" >> ./s2_logs/22DPMCERN/example1.log
```

Figure 8.4: S2 Example

S2 has allowed us to build a testing framework that supports the parallel execution of tests where the interactions among concurrent method invocations can be easily tested. The S2 test suite has allowed the early discovery of memory corruption problems by the CGSI security plug-ins in the CASTOR implementation. Authentication errors were reported randomly when multiple concurrent requests were made to the server. The coding of such a test case required very little time (few minutes) with no errors made in the test program that exercised this test case.

8.6 Conclusions: how to optimize test case design

The study of the SRM specification and the test case analysis and design have allowed us to drastically reduce the number of test cases to write, while keeping the test set pertinent with the SUT. From the methodologies applied and the results obtained we can deduce a testing strategy for the validation of a given implementation of a protocol with a specified interface. In what follows we list the main points:

- Analyze the specification and the user requirements and restrict as much as possible the number of input arguments or conditions for a method.
- Reduce to the minimum allowed number of possible fixed values that an input argument can assume, keeping the cases representative for the SUT. These types of arguments are the ones that determine a high number of test cases to be considered.

- Categorize the input conditions with respect to the output produced. Depending on such output limit even further the number of input conditions to consider.
- Use a mix of equivalence partitioning and boundary condition analysis for determining a basic set of tests that validate the basic behavior of a system with respect to the specifications.
- Study the type of the input and output arguments and their definition to design a set of test cases that check the response of the system to boundary conditions.
- Use cause-effect graphing to verify that the restrictions and conditions specified by the protocol are respected.
- Analyze the results of the tests designed so far to identify possible recurrent erroneous situations and use the error guessing methodology to develop further tests to discover basic implementation assumptions or design decisions not in accordance with the specifications.
- Study the interaction between all possible functions with the one targeted. Use UML activity diagrams to study and follow all possible interactions derived from the specifications.
- Use the cause-effect graphing methodology applied to the function interaction studies to systematically identify the test cases needed to validate the behavior of the SUT.
- Derive test cases from real use cases of the system usage.
- After designing the test cases derived so far, use the error guessing methodology and the experience gained to define more tests cases that can spot critical behaviors of the SUT.

The experience in running the tests has also shown other important points to consider while testing:

- It is very useful to have a completely implementation independent testing strategy. The SUT should be tested by people not involved in the development
- The results of the tests should be inspected fully even if the result is a success. Through this inspection it is possible to identify behaviors not initially foreseen and not taken into consideration.
- Tests should focus on valid and invalid behaviors.
- The probability of finding errors increases in areas where errors were previously found.
- The methodologies used have helped to identify a very important set of test cases. However, it is only the experience and the knowledge of the SUT that suggests how to strategically approach the testing design.
- Never change the testing suite to adapt to modified situations. Use the tests developed for the new situation to check if the response of the system is still the same (this was the case of the introduction of the new SRM 2.2 WSDL for the SRM implementations).
- Monitor the total number of test failed over the total number of tests executed over time for an implementation. Do not stop testing immediately when this number reaches zero, but wait until the system has completely stabilized the response.

Chapter 9

CONCLUSIONS

Storage management and access in a Grid environment is a rather new topic that is attracting the attention of many scientists. Given the heterogeneity of the existing storage solutions with their constant evolution and the complexity of the functionality required by Grid applications, a completely satisfactory solution to storage in the Grid has not yet been proposed. A particularly interesting Grid environment in terms of storage requirements is the WLCG, as we have seen in Chapter 3. Such requirements coming from the community of HEP experiments are rather complex, given the amount of data in the order of 15-20 Petabytes produced every year by the LHC accelerator. The WLCG infrastructure foresees a hierarchical organization of computing resource providers in tiers: CERN is the Tier-0 where data are produced; big computing centers able to handle Petabytes of data are the first level tiers or Tier-1s that hold copies of the data produced at CERN; Tier-2s are smaller centers generally responsible for the production of simulated data that are also stored at Tier-2s and Tier-1s; smaller tiers are the third level of the hierarchy where physics scientists have access to all sets of available data for analysis and results determination. The WLCG is a research infrastructure that poses interesting problems in terms of storage management and access:

- Data flow of significant amounts of data on a world-wide distributed network of resources.
- Uniform management for the persistent storage of data at the various centers deploying diverse storage solutions.
- Data access supporting a variety of data access and transfer protocols sometimes connected to legacy solutions.
- All this respecting the local access policies and security restrictions imposed by centers and the Virtual Organization involved in the process.

9.1 Standardizing the storage interface

In Chapter 4 we have classified the most used storage solutions adopted today in the centers part of the WLCG collaboration. We have also described the attempts made in industry to offer a transparent interface to storage at the block and file level via SAN or NAS infrastructures. Even if they offer the possibility of accessing the same files from many diverse OS implementations, these solutions are still proprietary and they do not offer a real transparent access from a world-wide distributed infrastructure such as the Grid.

The "Grid storage" initiative promoted by IBM and proposed in the Open Grid Forum's File System Working Group aims at creating a topology for scaling the capacity of NAS in response to application requirements, and introducing a technology for enabling and managing a single file system so that it can span an increasing volume of storage. In a Grid Storage NAS heads are joined together using clustering technology to create one virtual head and offer wide-area files sharing with virtualization services, file services and centralized management. Even though this project is a good step forward toward establishing a global Grid namespace for data and transparent wide-area network access to data, it does not solve the problem of offering a transparent interface to data stored on different media with different access latency and management technologies. Furthermore, it does not tackle the issue of proposing highly efficient data transfer tools based on protocols optimized for wide area network. Security, access policies and support

for legacy applications is also an area that is left to other OGF working groups to investigate, even if strongly connected to storage.

Another interesting project is the Internet Backplane Protocol (IBP) [87] mentioned in Chapter 6. It aims at providing a virtual publicly sharable storage all over the world. The L-Bone infrastructure today counts more than 580 IBP depots and serves more than 10s of Terabytes. The functionality offered by IBP is similar to the one offered by the SRM protocol. However, IBP is not only a control protocol for storage management but it also focuses on file access, imposing a specific protocol. The main idea is to make an online storage available for public usage. Therefore, quality of storage and transparent access to any kind of MSS is not normally a concern of IBP. Many projects have used IBP as a storage backbone to design other technologies. Among the others: DataTag, Condor, and the Network Weather Service to monitor the status of distributed computational resources. IBP introduces an original idea toward virtualization of storage resources.

In Chapter 6 we have introduced the SRM protocol proposed by the OGF SRM-WG that aims at proposing a control protocol for storage management and access in the Grid. The SRM protocol offers very important functionalities such as the possibility of dynamically reserving space of a certain quality, dynamic space and file management, access to system optimization patterns via the specification of data usage conditions (data processing conditions, WAN or LAN access, time of operations, etc.), the negotiation of file access and transfer protocols between client and server, possible optimized copy operations, uniform namespace management functions, file permission functions, transparent access guaranteed even for storage solutions with an implicit latency to data access, etc. We made several contributions to the SRM effort as pointed out in the various chapters of this thesis:

- A mathematical formalization for modeling the SRM protocol;
- Analysis of the consistency and completeness of the specifications;
- Modeling a standard test suite for the validation of an implementation.

Even though the SRM protocol is a real step forward toward the standardization of the storage control interface, the proposed v2.2 has shown already some of its limitations. The SRM 2.2 is about to enter the production phase in the WLCG infrastructure. The daily usage from the LHC Virtual Organizations will definitely provide the necessary input to the SRM definition process.

9.2 The SRM protocol: current limitations and open issues

While designing the SRM protocol, it was immediately obvious that it was not possible to accommodate from the beginning many of the requirements expressed by the various scientific communities. In fact, it was important to proceed immediately with implementing the interface in order to verify the response of the designed system to the requirements and the adaptability of the protocol to the existent MSS solutions used in WLCG and in other Grid initiatives. The cycle of designing a protocol with strong flexibility requirements must foresee a step-by-step approach where few functionalities are designed and implemented first and then verified against the requirements in order to understand if the approach followed is heading in the right direction or not.

The experience acquired with the design and implementation of SRM v2.2 has allowed us to acquire fundamental knowledge in terms of protocol design and specification techniques. As mentioned before, designing a protocol and achieving a wide acceptance is a long and time-consuming effort. In order to

make the SRM protocol even more effective and efficient for the HEP community, several additional features are required in the near future:

a) It must be possible for a VO manager to completely manage the space created for the members of the VO. Once a VO manager has created the space, he/she needs to be able to allow other VO members with the same or different VO roles to have equal rights on the space created so that a set of people can actually act on that space.
b) Furthermore, it must be possible for a normal user to allocate space in an area that the VO manager has previously allocated and to manage it, allowing access to other users.
c) Permissions on spaces should specify which individuals are allowed to perform which actions with a rather detailed granularity.
d) VO managers should be able to specify different priorities of operations for different groups in a virtual organization. In this case, if multiple groups ask for space resources, priority is given to those with a more important task to achieve while the others have to wait for resource availability. The same is valid for other operations that are normally queued by the underlying storage system.
e) Point d) falls under a wider requirement for an administrative interface to storage. Space in the Storage Class Tape0Disk1 is supposed to be managed by the VO. However, the SRM offers only limited tools to monitor the usage of resources, check at a given time who is the owner of the files or copies that reside in a given space and act on them resetting for instance pin or file lifetimes.
f) Another feature highly needed when dealing with a massive amount of data is the possibility of specifying true bulk operations on data. For instance, at the moment it is not possible to copy all files in a directory on a server to another directory on a remote server. The two servers involved in the operation can optimize the operation – which is usually not the case when working at the file-by-file level. For servers that do not have this capability, higher-level tools can be made available to optimize the job and perform bulk operations.
g) At the moment it is up to the user application to decide which space token to use for new files. It should be possible to allow the SRM to optimize and load balance the space usage. Clients should only deal with space token descriptions if they choose so, leaving to the underlying storage system an optimized management of the space.
h) At the moment no quota management is foreseen by the specification. This is not practical, since it is not possible at a central level to manage the space assigned to a VO and to impose upper level thresholds of usage. Also, for systems not supporting some kind of resource usage monitoring, it is easy for a user to exploit the system and use shared resources at his own advantage.
i) Other desired features are file locking mechanisms and the support for modifiable files. File locking mechanisms are needed by Grid services such as CONStanza [44], for the mirroring of modifiable files or databases over the Grid. In WLCG files are assumed to be read-only. Once they are created, they are un-modifiable. However, even in High Energy Physics, there are cases where it is convenient to allow for modifiable files. Modifications to SRM v2.2 have been proposed to accommodate the requests of the BaBar experiment to modify files created by several processes and to notify the SRM server when the file is finally committed. This can be done with the current version of the protocol. However, it should be possible to be able to query the TURL made available to a class of processes for put operations.

9.3 Toward SRM version 3

The SRM version 3 specifications are under discussion at the time of writing. The issues found and experienced during the work done on version 2 of the protocol are fundamental for the definition of

version 3. SRM version 3 will try to cover all issues reported in Section 9.2 and offer yet other features over version 2. Among the others we mention the following:

a) Support for hierarchical spaces.
b) A totally flexible management of files, associated paths, and spaces. Files and their copies can lie in some space and have an associated path. However, at any moment it will be possible to change the quality of space required for a file without changing his path.
c) The possibility to assign a priority to files so to distinguish between critical files over normal user files during SRM operations (purging files from spaces, moving files between spaces, etc.). Priorities can be established as well on operations.
d) The possibility for copies of a file to be in several spaces with several characteristics at a given time. Even though this feature is somehow present also in SRM v2.2, it has not being enforced in WLCG. However, in version 3 a new function will be added that will explicitly make it possible to add a copy of a file to a space with different characteristics with respect to the space where the master copy of a file was created.
e) SRM v3 allows for a better management of files', copies' and handles' lifetimes. Also, it will be possible to change the StorageType of a file.
f) Together with the srmBringOnline function, a similar function that allows files to be explicitly pushed in near line space will be made available.
g) The srmPrepareToPut will have an option that allows clients to specify when a file should be migrated to near line storage if that class of storage has been choosen, and if a copy should be left online or not.

9.4 Protocol validation

The design cycle of a protocol definition imposes its validation against available implementations. In Chapter 8 we have shown how important and complex this work is. In order to reduce the number of test cases to be considered, a very careful analysis of the user requirements and the working environment is necessary. The black box methodology was used to derive a persistent test set for the SRM SUTs available. We have seen how a single approach, out of the four covered by the black box methodology, is not enough in the SRM case to design a persistent testing suite. Therefore, a combination of equivalence partitioning, boundary condition analysis, cause-effect graphing and error guessing constitutes an efficient methodology in order to identify a rigorous test set for protocol implementation verification. In particular, we have applied equivalence partitioning, boundary condition analysis and cause-effect graphing to design a family of basic tests, to verify the most basic functionality of the protocol. The cause-effect graphing approach has allowed us to discover incompleteness and sometime incoherence in the protocol specification.

Cause-effect graphing was used as well in an unusual way to analyze the interactions between SRM calls and to examine the effect of concurrency. It has allowed for the discovery of synchronization problems between the SRM front-end server and the MSS storage servers in some cases. The cause-effect graphing was used together with other black box approaches to design the use-case SRM test suite, a test case family created to investigate deeper the functionalities offered by the SRM implementations. After performing these two fundamental steps, the family of test suite was enlarged to include the design of interoperability, exhaustive and stress test suites to exercise interoperability functions between SRM servers, the format of input and output arguments and to stress the implementations with concurrent and contradictory requests. From this exercise we have deduced a set of general rules to keep into consideration when testing a protocol.

9.5 Future Work

Even if the research in the area of storage access and management in a Grid environment has made a lot of progress recently, we are still far from a protocol definition that makes access to storage completely uniform and flexible. Making the proposed SRM a real standard will certainly help in this direction. However, we probably need to consider the complexity of the protocol as it is today and see if it is really necessary to expose that complexity to client applications. Furthermore, the integration of such a protocol with other Grid services is an area of research that needs to be considered and widened. For instance, it is interesting to understand how the work that is advancing in the area of "Grid storage" for a global data namespace can be interfaced to the SRM. As a last note, the technology for the verification of a protocol needs to be further developed and tested on real grounds, such as that offered by the WLCG infrastructure. The test modeling theory could probably help significantly.

BIBLIOGRAPHY

[1] I. Foster, K. Kesselman *The Grid: Blueprint for a New Computing Infrastructure*, 1st ed. Morgan Kaufmann Publisher, ISBN: 1558604758, November 1998.

[2] H. Stockinger, F. Donno et al. *Defining the Grid: a Snapshot on the Current View*, submitted to the Journal of SuperComputing, June 2006.

[3] T. Weishäupl, F. Donno, E. Schikuta, H. Stockinger, H. Wanek *Business In the Grid: The BIG Project*, 2nd International Workshop on Grid Economics and Business Models (GECON2005), Seoul, Korea, March 13, 2005.

[4] I. Foster, C. Kesselman *Computational Grids*, Chapter 2 of "The Grid: Blueprint for a New Computing Infrastructure", Morgan-Kaufman, 1999.

[5] F. Donno et al. *Replica Management in the EU DataGrid Project*, International Journal of Grid Computing,), ISSN 1570-7873.

[6] F. Donno, H. Stockinger, R. Puccinelli, K. Stockinger *Data Grid Tutorials with Hands-on Experience,* 1st International Workshop on Grid Education (Grid.Edu 2004), Chicago, Illinois, April 19-22, 2004, IEEE Computer Society Press;

[7] F. Donno et al. *The INFN-GRID Testbed*, Future Generation Computer System in Advanced Grid Technologies, Elsevier, Special Issue, Volume 21, Issue 2, 1 February 2005, Pages 249-258

[8] F. Donno, S. Fantinel, D. Rebatto, G. Tortone, L. Vaccarossa, M. Verlato *The WorldGrid transatlantic testbed: a successful example of Grid interoperability across EU and US domains*, CHEP2003 March 24-28, 2003 La Jolla, California;

[9] E. Schikuta, F. Donno, H. Stockinger, E. Vinek, H. Wanek, T. Weishäupl, C. Witzany *Business In the Grid: Project Results*, 1st Austrian Grid Symposium, OCG Verlag, Hagenberg, Austria, December 1-2, 2005.

[10] F. Donno, L. Gaido, A. Ghiselli, F. Prelz, M. Sgaravatto *DataGrid Prototype 1*, Terena Networking Conference 2002, Limerick - Ireland, 3 - 6 June 2002

[11] I. Foster, C. Kesselman, S. Tuecke *The Anatomy of the Grid: Enabling Scalable Virtual Organizations,* International Journal of Supercomputer Applications, 15(3). 2001.

[12] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke *The Physiology of the Grid: An Open Grid Services Architecture for Distributed System Integration*, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[13] S. Campana, A. Delgado Peris, F. Donno, P. Mendez Lorenzo, R. Santinelli, A. Sciaba' *Toward a Grid Technology Independent Programming Interface for HEP Applications*, CHEP2004 Sept.27- Oct.1, 2004 Interlaken, Switzerland

[14] The Globus Project Web Site, http://www.globus.org

[15] L. Srinivasan, J. Treadwell *An Overview of Service-oriented Architecture, Web Services and Grid Computing*, HP Software Global Business Unit, November 3, 2005

[16] European Data Grid Project Web Site, http://cern.ch/eu-datagrid

[17] Worldwide LHC Computing Grid Project Web Site, http://www.cern.ch/lcg.

[18] NorduGrid Project Web Site, http://www.nordugrid.org

[19] Open Science Grid Project Web Site, http://www.opensciencegrid.org

[20] Load Sharing Facility, http://en.wikipedia.org/wiki/Platform_LSF

[21] Teragrid Project Web Site, http://www.teragrid.org

[22] Enabling Grid for E-SciencE Project Web Site, http://www.eu-egee.org

[23] Kerberos authentication: http://web.mit.edu/Kerberos/

[24] S. Campana, A. Delgado Peris, F. Donno, P. Mendez Lorenzo, R. Santinelli, A. Sciaba' *Experience integrating a General Information System API in LCG Job Management and Monitoring Services*, CHEP2004 Sept.27- Oct.1, 2004 Interlaken, Switzerland

[25] F. Donno et al. *Report on the INFNGRID Globus Evaluation* CHEP'01, September 3-7, 2001 Beijing, China.

[26] LDAP: http://en.wikipedia.org/wiki/LDAP
[27] GLUE in OGF: www-unix.mcs.anl.gov/gridforum/gis
[28] Top500 Supercomputers web site: http://www.top500.org/
[29] CoreGrid web site: http://www.coregrid.net
[30] OGF web site: http://www.ggf.org/
[31] F. Berman and R. Wolski *The AppLeS Project: A Status Report*, Proc. of the 8th NEC Research Symposium, Berlin, Germany, May 1997.
[32] H. Nakada, M. Sato, and S. Sekiguchi *Design and Implementation of Ninf: towards a Global Computing Infrastructure*, Future Generation Computing Systems, October 1999. Special Issue on Metacomputing
[33] H. Casanova and J. Dongarra *NetSolve: A Network Server for Solving Computational Science Problems*, Intl. Journal of Supercomputing Applications and High Performance Computing, 11(3), 1997.
[34] S. Bagnasco, P. Cerello, R. Barbera, P. Buncic, F. Carminati, P. Saiz *AliEn, EDG Interoperability in Alice*, CHEP2003, La Jolla, 24-28 March 2003
[35] P. Chandra, A. Fisher, and C. Kosak et al. *Darwin: Customizable Resource Management for Value-Added Network Services*, 6 th Int. Conf. on Network Protocols. IEEE,1988
[36] R. Buyya, D. Abramson, and J. Giddy *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*, Int. Conf. on High Performance Computing in Asia-Pacific Region, Beijing, China, 2000. IEEE-CS Press.
[37] F. Donno, A. Fanfani, F. Fanzago, V. Garbellotto, L. Vaccarossa, M. Verlato *Atlas and CMS applications on the WorldGRID testbed*, CHEP2003 March 24-28, 2003 La Jolla, California;
[38] H. Stockinger, F. Donno, E. Laure, S. Muzaffar, G. Andronico, P. Kunszt, P. Millar *Grid Data Management in Action: Experience in Running and Supporting Data Management Services in the EU DataGrid Project*, CHEP2003 March 24-28, 2003 La Jolla, California;
[39] D. Cameron, J. Casey, L. Guy, P. Kunszt, S. Lemaitre, G. McCance, H. Stockinger, K. Stockinger, G. Andronico, W. Bell, I. Ben-Akiva, D. Bosio, R. Chytracek, A. Domenici, F. Donno, W. Hoschek, E. Laure, L. Lucio, P. Millar, L. Salconi, B. Segal, M. Silander *Replica Management in the EU DataGrid Project*, International Journal of Grid Computing (2004), ISSN 1570-7873
[40] D. Cameron, J. Casey, L. Guy, P. Kunszt, S. Lemaitre, G. McCance, H. Stockinger, K. Stockinger, G. Andronico, W. Bell, I. Ben-Akiva, D. Bosio, R. Chytracek, A. Domenici, F. Donno, W. Hoschek, E. Laure, L. Lucio, P. Millar, L. Salconi, B. Segal, M. Silander *Replica Management Services in the European DataGrid Project*, UK e-Science Programme All Hands Meeting (AHM 2004) Nottingham, September 2004
[41] A. Domenici, F. Donno, G. Pucciani, H. Stockinger, K. Stockinger *Replica Consistency in a Data Grid*, IX International Workshop on Advanced Computing and Analysis Techniques in Physics Research December 1-5, 2003 High Energy Accelerator Research Organization (KEK) 1-1 Oho, Tsukuba, Ibaraki 305-0801 Japan
[42] H. Stockinger, A. Samar, S. Muzaffar, F. Donno, A. Domenici *Grid Data Mirroring Package (GDMP)*, Global Grid Forum 3, Frascati, Italy, 7-10 October 2001.
[43] H. Stockinger, A. Samar, S. Muzafar, F. Donno *Grid Data Mirroring Package (GDMP)*, Scientific Programming Journal, Special Issue devoted to Grid Computing, 2002.
[44] A. Domenici, F. Donno, G. Pucciani, H. Stockinger *Relaxed Data Consistency with CONStanza*, 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid2006), IEEE Computer Press, Singapore, May 16-19, 2006.
[45] R. Alfieri, F. Spataro, C. Anglano, R. Barbera, C. Rocca, M. Biasotto, P. Cerello, A. Forte, L. Gaido, A. Guarise, S. Lusso, A. Chierici, T. Ferrari, L. Fonti, F. Giacomini, G. Vita Finzi, A. Controzzi, F. Donno, A. Sciaba, Z. Xie, A. Gianoli, L. Marzola, C. Grandi, F. Semeria, I. Lippi, M. Sgaravatto, G. Lo Biondo, F. Prelz, S. Resconi, G. Tortone *Report on the INFN-GRID Globus evaluation* June 30, 2001; INFN/TC-02-30
[46] NorduGrid: http://www.nordugrid.org

[47] A. S. Grimshaw, A. Natrajan, M. A. Humphrey, M. J. Lewis, A. Nguyen-Tuong, J. F. Karpovich, M. M. Morgan, A. J. Ferrari *From Legion to Avaki: The Persistence of Vision*, chapter 32, Grid Computing, 29 May 2003, ISBN: 0470853190, Editor(s): F. Berman, G. Fox, T. Hey

[48] Nordic Data Grid Facility web site: http://www.ndgf.org/

[49] gLite middleware: http://glite.web.cern.ch/

[50] WLCG Site Functional Test web page: https://lcg-sft.cern.ch/sft/lastreport.cgi

[51] S. Burke, S. Campana, A. Delgado Peris, F. Donno, P. Mendez Lorenzo, R. Santinelli, A. Sciaba' *gLite 3.0 User Guide – WLCG Manual Series*, 16 May 2006

[52] DataTAG – Research & technological development for a Data TransAtlantic Grid http://cern.ch/datatag/

[53] GriPhyN – Grid Physics Network http://www.griphyn.org/

[54] iVDgL – International Virtual Data Grid Laboratory http://www.ivdgl.org/

[55] The Virtual Data Toolkit: http://vdt.cs.wisc.edu/

[56] The Condor Project: http://www.cs.wisc.edu/condor/

[57] J. Basney *MyProxy Protocol* Global Grid Forum Experimental Document GFD-E.54, November 26, 2005.

[58] Grid Monitoring Architecture (GMA), OGF Working Group: http://www-didc.lbl.gov/GGF-PERF/GMA-WG/

[59] M. Corbatto *An introduction to Portable Batch System (PBS)*, January 2000 http://hpc.sissa.it/pbs/pbs.html

[60] The Storage Resource Manager Working Group: http://sdm.lbl.gov/srm-wg/

[61] Globus Community Authorization Service: http://www-unix.globus.org/toolkit/docs/4.0/security/cas/

[62] R-GMA: Relational Grid Monitoring Architecture: http://www.r-gma.org/

[63] H. Stockinger, F. Donno, G. Eulisse, M. Mazzucato, C. Steenbert *Matchmaking, Datasets and Physics Analysis* Workshop on Web and Grid Services for Scientific Data Analysis (WAGSSDA) 2005 International Conference on Parallel Processing (ICPP-2005), Oslo, Norway

[64] M. Ernst, P. Fuhrmann, T. Mkrtchyan, J. Bakken, I. Fisk, T. Perelmutov, D. Petravick, *Managed data storage and data access services for Data Grids* CHEP, La Jolla, California, March 2004

[65] CASTOR: CERN Advanced STORage manager - http://castor.web.cern.ch/castor

[66] LCG Disk Pool Manager (DPM): https://twiki.cern.ch/twiki/bin/view/LCG/DpmAdminGuide

[67] Alice Collaboration *Alice Technical Design Report of the Computing* CERN-LHCC-2005-018, ALICE, TDR-012, 15 June 2005: http://aliceinfo.cern.ch/static/Documents/TDR/Computing/All/alice_computing.pdf

[68] ATLAS Collaboration *ATLAS Computing Technical Design Report* CERN-LHCC-2005-017, ATLAS, TDR-017, 20 June 2005: http://cern.ch/atlas-proj-computing-tdr/PDF/Computing-TDR-final-June20.pdf

[69] CMS Collaboration *CMS Computing Technical Design Report* CERN-LHCC-2005-023, CMS, TDR-007 http://cdsweb.cern.ch/search?id=838359

[70] LHCb Collaboration *LHCb Computing Technical Design Report* CERN-LHCC-2005-019, LHCb, TDR-019, 20 June 2005: http://doc.cern.ch/archive/electronic/cern/preprints/lhcc/public/lhcc-2005-019.pdf

[71] F. Donno *YASS, Yet Another Simple Stager: Motivations and Design*, INFN-Pisa Internal Notes – INFNPI1002012000, January 2 2000

[72] M. Luvisetto, P. Veronesi, *Spec values and normalization criterions* INFNGRID20051118-1330, v1.0.1, 27 January 2006

[73] ROOT, An Object Oriented Data Analysis framework: http://root.cern.ch/

[74] The xrootd software: http://xrootd.slac.stanford.edu/

[75] The Grid File Access Library: http://grid-deployment.web.cern.ch/grid-deployment/gis/GFAL/gfal.3.html

[76] F. Donno et al. *The WLCG Baseline Service Working Group* http://cern.ch/lcg/PEB/BS

[77] F. Donno et al. *The WLCG Baseline Service Working Group Report* v1.0, 24 June 2005 http://lcg.web.cern.ch/LCG/peb/bs/BSReport-v1.0.pdf

[78] F. Donno et al The WLCG Storage Class Working Group: http://cern.ch/LCG/SRMdev

[79] S. Farrell, R. Housley *An Internet Attribute Certificate Profile for Authorization* RFC 3281

[80] V. Ciaschini, A. Frohner *Voms Credential Format* http://edgwp2.web.cern.ch/edg-wp2/security/voms/edg-voms-credential.pdf

[81] A. Shoshani, P. Kunszt, H. Stockinger, K. Stockinger, E. Laure, J.-P. Baud, J. Jensen, E. Knezo, S. Occhetti, O. Wynge, O. Barring, B. Hess, A. Kowalski, C. Watson, D. Petravick, T. Perelmutov, R. Wellner, J. Gu, A. Sim *Storage Resource Management: Concepts, Functionality, and Interface Specification*, GGF 10, The Future of Grid Data Environment, 9-13 March 2004, Humboldt University, Berlin Germany

[82] A. Sim, F. Donno and all, *SRM v2.2 Specification*, 15 December 2006, http://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.html

[83] F. Donno et al., *SRM v2.2 WSDL Interface*, 15 December 2006, http://sdm.lbl.gov/srm-wg/srm.v2.2.wsdl

[84] *RFIO at IN2P3*, http://cc.in2p3.fr/docenligne/289, November 2006

[85] C. Boeheim, A. Hanushevsky, D. Leith, R. Melen, R. Mount, T. Pulliam, B. Weeks *Scalla: Scalable Cluster Architecture for Low Latency Access Using xrootd and olbd Servers* 22 August 2006, http://xrootd.slac.stanford.edu/

[86] A. Hanushevsky The xrootd Protocol, Version 2.4.5 16 Augusut 2005, http://xrootd.slac.standford.edu/doc/XRdv245/XRdv245.htm

[87] J. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski *The Internet Backplane Protocol: Storage in the network* in Network Storage Symposium, 1999.

[88] GLUE Schema version 1.2: http://glueschema.forge.cnaf.infn.it/V12

[89] P. Badino, J.P. Baud, S. Burke, E. Corso, S. De Witt, F. Donno, P. Fuhrmann, M. Litmaath, R. Zappi *Storage Element Model for SRM 2.2 and GLUE schema description* Internal WLCG technical report, v3.5, 27 October 2006

[90] M. Farley *Storage Networking Fundamentals* Cisco Press; ISBN: 1587051621, Dec. 2004.

[91] ESCON: http://www.c2p.com/ESCONTraining.pdf , June 2005.

[92] H. Meinhard *Disk storage technology for the LHC T0/T1 centre at CERN* CHEP2004 Sept.27- Oct.1, 2004 Interlaken, Switzerland

[93] G. J. Myers, C. Sandler (Revised by), T. Badgett (Revised by), T. M. Thomas (Revised by) *The ART of SOFTWARE TESTING 2nd edition*, December 2004, ISBN 0-471-46912-2

[94] L. Lucio *Theory of Testing* Lectures in Computer Science, Summer Term 2006, University of Geneva, Switzerland

[95] A. Pretschner, J. Philipps *Methodological Issues in Model-Based Testing* (chapter 10 of Model-Based Testing of Reactive Systems), July 2005, ISBN 978-3-540-26278-7

[96] F. Donno *Open issues in SRM 2.2*, 2 February 2007. https://twiki.cern.ch/twiki/bin/view/SRMDev/

[97] F. Donno, J. Menjak *The S2 testing suite* 15 September 2006, http://s-2.sourceforge.net

[98] *Fiber Channel: Overview of the technology* http://www.fibrechannel.org/technology/overview.html

[99] D. Bonacorsi *CMS at CNAF Tier1: Pre-challenge production and Data Challenge 04*, III Workshop Calcolo-Reti INFN, Castiadas (CA) - May 24-28, 2004. http://www.ca.infn.it/ws2004/doc/III_giornata/bonacorsi.ppt

[100] J.-P. Baud, J. Casey *Evolution of LCG-2 Data Management* CHEP, La Jolla, California, March 2004.

[101] F. Donno *Enstore Systen Analysis for Data Handling in CDF Run II*, CDF Publications, FERMILAB, CDF Note no. 4775 (pp 10), October 12, 1998.

[102] V. Ciaschini, A. Ferraro, A. Ghiselli, G. Rubini, R. Zappi, A. Caltroni *G-PBox: A Policy Framework for Grid Environments* CHEP04, Sept.27- Oct.1, 2004 Interlaken, Switzerland

[103] J. Bent, V. Venkataramani, N. LeRoy, A. Roy, J. Stanley, A. C. Arpaci Dusseau, R. H. Arpaci-Dusseau, M. Livny *Flexibility, Manageability, and Performance in a Grid Storage Appliance*, 11th IEEE Symposium on High Performance Distributed Computing, Edinburgh, Scotland, July 2002.

[104] A. Shoshani, A. Sim, J. Gu *Storage Resource Managers: Middleware Components for Grid Storage*, 9th IEEE Symposium on Mass Storage Systems, 2002.

[105] *SAM* http://d0db.fnal.gov/sam, Jan 2007

[106] HPSS http://www.hpss-collaboration.org/hpss/index.jsp, June 2005

[107] TSM : http://www-306.ibm.com/software/tivoli/products/storage-mgr/, Jan. 2007.

[108] J. Wolfrat, P. de Boer, W. de Jong, R. Trompert. *A Mass storage solution for Grid environments* EUNIS 2003.

[109] *SGI's CXFS file system*, http://www.sgi.com/products/storage/tech/file_systems.html, Jan. 2007.

[110] SRB: http://www.sdsc.edu/srb/, June 2005.

[111] D. Hildebrand, P. Honeyman *Exporting Storage Systems in a Scalable Manner with pNFS* 22nd IEEE and 13th NASA Goddard Conference on Mass Storage Systems and Technologies, Monterey, CA, April 2005.

[112] T. Olivares, L.Orozco-Barbosa, F.Quiles, A. Garrido, P.J.Garcia *Performance study of NFS over Myrinet-based clusters for parallel multimedia applications* 2001 IEEE Canadian Conference on Electrical and Computer Engineering- CCECE, Toronto (Ontario), Canada, May 13-16, 2001.

[113] S. Blumson *AFS write performance – A campaign paper*, CITI Technical Report, July 1992, http://www.citi.umich.edu/techreports/reports/citi-tr-92-7.pdf

[114] GGF – GFS WG: https://forge.gridforum.org/projects/gfs-wg/, Jan. 2007.

[115] F. Schmuck, R. Haskin *GPFS: A Shared-Disk File System for Large Computing Clusters*, Conference on File And Storage Technologies, Monterey, CA, January 28-30, 2002

[116] Lustre: A Scalable, High-Performance File System White Paper, Cluster File Systems, Inc., May 2006, http://www.lustre.org/docs/whitepaper.pdf

[117] Posix 1003.1e/1003.2c Draft Standard 17 (withdrawn) http://wt.xpilot.org/publications/posix.1e/

[118] *System Management: Data Storage Management (XDSM) API Common Application Environment (CAE) Specification C429* The Open Group, ISBN 1-85912-190-X

[119] A. Shoshani, L. M. Bernardo, H. Nordberg, D. Rotem, A. Sim *Multidimensional Indexing and Query Coordination for Tertiary Storage Management* Proceedings of the 11th International Conference on Scientific and Statistical Database Management (SSDBM'99).

[120] *SRM v2.2 WLCG usage agreement* Fermilab, Grid Storage Interfaces Workshop, May 2006 http://cd-docdb.fnal.gov/0015/001583/001/SRMLCG-MoU-day2%5B1%5D.pdf

[121] F. Donno *SRM v2.2 – Issues with the specifications* CERN, 20 Feb 2007 https://twiki.cern.ch/twiki/bin/view/SRMDev/